AFRL-IF-RS-TR-2000-33
Final Technical Report
MARCH 2000

DECISION SUPPORT TOOLS FOR AIRCRAFT OPERATIONS PLANNING WITH MANY PRACTICAL CONSTRAINTS

Rutgers University Operations Research Center

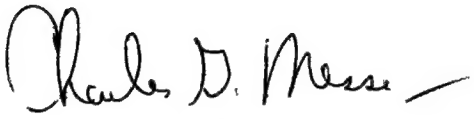
Lei Lei

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2000-33 has been reviewed and is approved for publication.

APPROVED: 

CHARLES G. MESSENGER
Project Engineer

FOR THE DIRECTOR: 

NORTHRUP FOWLER, Technical Advisor
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2000	Final Feb 97 - Jul 99

4. TITLE AND SUBTITLE
DECISION SUPPORT TOOLS FOR AIRCRAFT OPERATIONS PLANNING WITH MANY PRACTICAL CONSTRAINTS

5. FUNDING NUMBERS
C - F30602-97-1-0043
PE - 61102F
PR - 2304
TA - OR
WA - P3

6. AUTHOR(S)
Lei Lei

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rutgers University Operations Research Center
Room 2/3, Ackerson Hall
180 University Avenue
Newark NH 07102

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-33

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Charles G. Messenger/IFTD/(315) 330-3528

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this project, decision support tools for aircraft operations planning under many practical constraints were developed. The application that motivated this study was the aircraft planning and logistics planning and scheduling that occurred during the planning of operations such as Desert Storm and the Somalia effort. This report describes a family of mathematical models and computation procedures that can be used as decision support tools for aircraft operations planning with many practical constraints. The major advantage of this planning tool is that it is able to find the minimum-cost plan under many complex constraints without the need to solve any mathematical program. The resulting planning process is an iterative process.

14. SUBJECT TERMS
Decision Support Tools, Planned, Scheduling, Aircraft Operations

15. NUMBER OF PAGES
72

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

Abstract

We report a family of mathematical models and computation procedures that can be used as decision support tools for aircraft operations planning with many practical constraints. Our main results include:

1. A column generation model for the general aircraft operations planning problem. This model can be used to solve problems of relatively smaller sizes, or be modified as a decision tool to be used as part of an AI based approach to the planning problem. The resulting solution process is an iterative process, and each iteration identifies a set of candidate tours to be assigned to aircraft.

2. An integer program for the optimal matching of aircraft types and move types. While this model does not consider the traveling time, the resulting solution specifies the way of assigning moves to aircraft to maximize the aircraft capacity utilization and provides useful information for aircraft capacity planning. This model is efficient since the resulting mathematical programming model is decomposable and is independent of the number of moves.

3. A computation procedure that determines the minimum fleet size required to deliver a given set of moves within the specified time windows, assuming all the aircraft are identical and each aircraft leaves the destination of a move exactly at the due time (or due time plus a constant ground time). For the applications where aircraft may leave earlier than the due time of a move, the result estimates an upper bound on the fleet size needed. The resulting algorithm is a strongly polynomial algorithm and can be implemented to handle planning problems with large number of aircraft easily. A detailed outline for the algorithm is given.

4. A computation procedure that generates alternative feasible routes for aircraft in a general operations planning problem. It can be used independently as a planning

method for determining the most utilized route for each aircraft, or be used as a decision support tool for generating alternative feasible tours that satisfies the constraints. The resulting algorithm is easy to implement. Pseudo code is provided.

5. A sequential assignment method that finds the minimum-cost operations plan for a given fleet of aircraft of different types and a set of moves with different priorities and demand. The major advantage of this planning tool is that it is able to find the minimum-cost plan under many complex constraints without the need to solve any mathematical program. The resulting planning process is an iterative process. In each iteration a subset of moves are selected and added to the existing routes of respective aircraft so that the cost is minimized. This planning tool has been tested on computer and runs very fast. The computer program source code (in C++) is provided in the Appendix of this report. The format of required input data files is specified in section 8.3.

1. Introduction

In this project, we develop decision support tools for aircraft operations planning under many practical constraints. The application that motivated this study was the aircraft planning problem and the logistics planning and scheduling that occurred during the planning of operations such as Desert Storm and the Somalia effort.

The general aircraft operations planning problem can be stated as follows:

We are given a set of transportation operations, called "moves", where each move has a specified release time, due time, origin location, destination location, and consists of cargo of different types (e.g., passengers, oversize cargo, bulk cargo, outsize cargo, etc.) with respective quantities. We are also given a set of aircraft. Aircraft are heterogeneous in terms of their own maximum capacity for each cargo type, required ground time, maximum flying hours, traveling speed, and home base. In addition

- An aircraft may carry the cargo from multiple moves at the same time. This means that an aircraft arriving at the origin of a move may not have to be empty.

- A move may not get a direct flight. It may stay on an aircraft for many intermediate stops (since the aircraft may service multiple moves at the same time). Furthermore, a move may switch between aircraft. A move may be transported, for example, from New York to Chicago by one aircraft and then from Chicago to Seoul by another aircraft.

- The load (i.e., quantity of cargo) of a move may be greater than the maximum capacities (with respect to the cargo types) of an aircraft. This means that an aircraft may have to travel back and force multiple times to service a single move, or multiple aircraft may have to be assigned to serve a single move.

• Moves are not available for pick up before their release time, and must be delivered to their respective destinations before the due time. In some cases, a move may arrive late at the expenses of a penalty.

A small sample of moves and aircraft types, using the data format of TPFDD (Time-Phased Force Deployment Data) specified by the US Transportation Command, is given in Tables 1-1 and 1-2 below.

Table 1-1. A sample of moves

Move Type	T1A	T1AA	T1ABP	T1ABC	T1AC	T1AD
Passenger	212	77	474	0	41	197
Bulk	101	166	0	157	107	350
Oversize	2002	722	0	632	0	1978
Outsize	0	0	0	41	0	138
Origin	PTFL	SBEA	PTFL	PTFL	DKFX	SBEA
Destination	MEPJ	MEPJ	MEPJ	MEPJ	SMYU	SMYU
Release Time	C010	C001	C001	C002	C007	C003
Due Time	C012	C003	C002	C005	C009	C012

Table 1-2. A sample of aircraft types

Aircraft type	B747	LRwp	DC8	C5	DC 1030	C141	KC10	C130	LRwc	DC10 30CF	DC8 73CF
Passenger	401	401	160	73	30	22	16	8	0	0	0
Bulk	0	0	0	83	0	30	53	14	90	74	52
Oversize	0	0	0	72	0	30	53	13	90	37	0
Outsize	0	0	0	78	0	0	0	0	0	0	0

The common objectives for operations planning are typically to minimize the required fleet size for a given set of moves (with no delay allowed) or to minimize the resulting delay for a given fleet size. The two versions of the planning problem are closely related.

The resulting operations planning problem is a very difficult one. Its complexity often makes an "obvious" plan, produced by a simple or straight forward manual process, a low quality solution. To understand this issue, let's look at the following example:

Example. Consider that we are given two aircraft: DC8 with a capacity for 160 passengers, and C141 with a capacity for 22 passengers and 30 units of oversize cargo.

Both are home based at Boston (see Figure 1-1). There are two moves to be served. Move A has 150 passengers heading from New York to Honolulu, will be released at time t=0 and due at time t=48. Move B consists of 62 passengers and 27 units of oversize cargo heading from Chicago to Hilo, will be released at time t=3 and due at time t=21.

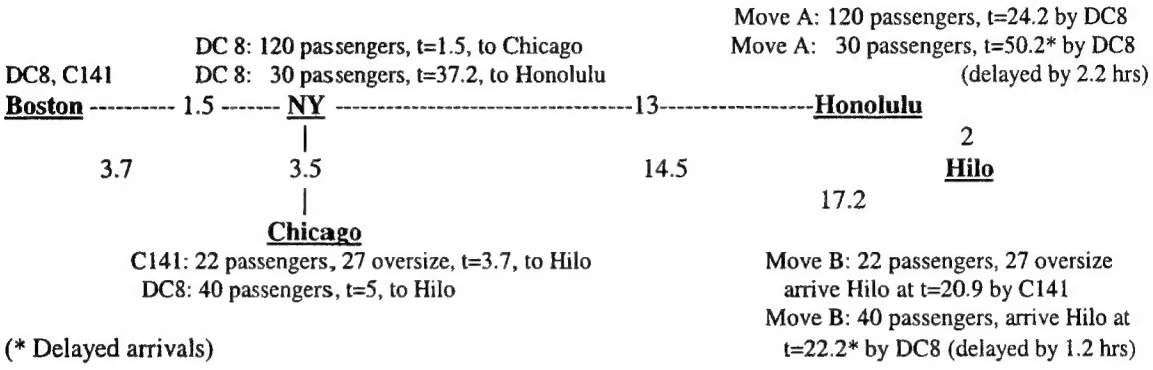


Figure 1-1. An example of operations plan with two moves and two aircraft, where the number of arc between cities indicates the required traveling time and ground time.

The optimal plan for this simple example is shown in Figure 1-1, which gives a total weighted delay

$$g = \sum_{\forall moves} Quantity\,being\,delayed \cdot tardiness \quad = 40\text{x}1.2+30\text{x}2.2=114$$

This optimal plan assigns two trips of DC8 to move A, and splits move B between DC8 and C141. Also, it requires 120 passengers on DC8 to experience two extra stops: Chicago and Hilo. On the other hand, an obvious plan in this case would be:

C141: Go to Chicago, pick up 22 passengers (the maximum capacity of C141) and 27 units of oversize cargo, and then go to Hilo;

DC8: Go to New York, pick up 150 passengers (the maximum demand of move A), go to Honolulu, drop off the passengers, go to Chicago, pick up the remaining 40 passengers, and then go to Hilo.

This plan, however, will end up with a total weighted delay g = 40x25.2=1008.

This is a very simple example involving only two moves and two aircraft. In reality, a typical problem would easily come up with 10,000 to 20,000 moves and 50-300 aircraft. This means that we need effective decision support tools that are able to find or to assist getting a feasible and quality operations plan within reasonable time.

Motivated by this need, we develop in this project a set of mathematical models and computation procedures that can be applied to quickly solve, either optimally or heuristically, different versions of the aircraft operations planning problem. They are:

1. *A column generation model for the general aircraft operations planning problem*

   This model can be used to solve problems of relatively smaller sizes, or be modified as a decision tool to be used as part of an AI based approach to the planning problem. The resulting solution process is an iterative process, and each iteration identifies a set of candidate tours to be assigned to aircraft.

2. *An integer program for the optimal matching of aircraft types and move types*

   While this model does not consider the traveling time, the resulting solution specifies the way of assigning moves to aircraft to maximize the aircraft capacity utilization and provides useful information for aircraft capacity planning. This model is efficient since the resulting mathematical programming model is decomposable and is independent of the number of moves.

3. *A computation procedure that determines the minimum fleet size required to deliver a given set of moves within the specified time windows, assuming all the aircraft are identical and each aircraft leaves the destination of a move exactly at the due time (or due time plus a constant ground time)*

   For the applications where aircraft may leave earlier than the due time of a move, the result estimates an upper bound on the fleet size needed. The resulting algorithm is a strongly polynomial algorithm and can be implemented to handle planning problems with large number of aircraft easily. A detailed outline for the algorithm is given.

4. *A computation procedure that generates alternative feasible routes for aircraft in general operations planning problem*

It can be used independently as a planning method for determining the most utilized route for each aircraft, or be used as a decision support tool for generating alternative feasible tours that satisfies the constraints. The resulting algorithm is easy to implement. Pseudo code is provided.

5. *A sequential assignment method that finds the minimum-cost operations plan for a given fleet of aircraft of different types and a set of moves with different priorities and Demand*

The major advantage of this planning tool is that it is able to find the minimum-cost plan under many complex constraints without the need to solve any mathematical program. The resulting planning process is an iterative process. In each iteration, a subset of moves are selected and added to the existing routes of respective aircraft so that the cost is minimized. This planning tool has been tested on computer and runs very fast. The computer program source code (in C++) is provided in the Appendix of this report. The format of required input data files is specified in section 8.3.

2. A mathematical formulation for the aircraft planning problem

In this section, we give a mathematical formulation for the aircraft operations planning problem to be studied in this project. This formulation assumes that once a move, or part of a move, is picked up by an aircraft, it stays on that aircraft until it arrives at its destination. That is, we exclude the case that a move can switch from one aircraft to another during its transportation.

Notation:

K: Total number of cargo types, k =1,2,…,K;

M: The set of moves, m =1,2,…,M;

J: The set of aircraft, f =1,2,…,$\|J\|$;

move m: $[o_m, g_m, r_m, d_m, q_{mk}, k=1,2,\ldots,K]$, where for any $m \in M$

- o_m: Origin of move m;
- g_m: Destination of move m;
- r_m: Release time of move m;
- d_m: Due time of move m;
- q_{mk}: Quantity of cargo k in move m;

aircraft f: $[a_f, b_f, D_f, Q_{fk}, k=1,2,\ldots,K]$, where for any $f \in J$

- a_f: Beginning time of the service window of aircraft f;
- b_f: Ending time of the service window of aircraft f;
- D_f: Depot (home base) of f;
- Q_{fk}: maximum capacity for carrying cargo of type k;

S^+/S^-: set of origins/destinations;

$X_{i,j,f}$ = 1 if f flies directly from location i to location j;
0 otherwise;

$Z_{m,k,f,j}$: Amount of cargo k of move m assigned to aircraft f in its j-th trip for move m;

Y_j: Departure time of an aircraft from location j;

P_f = $[p_f[0], p_f[1],\ldots, p_f[n_f]]$, a possible route of aircraft f, where $p_f[j]$ denotes the location of the j-th stop, and n_f denotes the total number of stops made by aircraft f before it returns to D_f, and

$$X_{p[j],\, p[j+1],\, f} = 1, \text{ for } j=1, 2, \ldots, n_f-1;$$

$t_{i,j,f}$ = Required traveling time from location i to location j by aircraft f (which may also include the ground time at location j).

Objectives:

G1). minimize the total number of aircraft

or G2). minimize the total delay to moves

Subject to the following constraints

1). A move must be completed

$$\sum_{\forall f}\sum_{\forall j} Z_{mkfj} = q_{mk} \qquad m = 1,2,...,M,\ k = 1,2,...,K$$

2). Flow conservation must be followed

$$\sum_{\forall i} X_{ijf} = \sum_{\forall i} X_{jif} \qquad \textit{for all non-home base locations}$$

3). All dispatched aircraft must return to their home bases

$$\sum_{\forall i} X_{D_f,i,f} = \sum_{\forall i} X_{i,D_f,f} \le 1 \qquad f = 1,2,...,\| J \|$$

4). Time window constraints on moves

$$Z_{mkfj} > 0 \text{ and } o_m = P_f[l] \Rightarrow Y_{P_f[l]} \ge r_m$$

$$Z_{mkfj} > 0 \text{ and } g_m = P_f[l] \Rightarrow Y_{P_f[l-1]} + t_{P_f[l-1],P_f[l],f} \le d_m$$

$m=1,2,...,M$

where the second set of constraints are relaxed for G2

5). Flying time constraints

$$Y_{P_f[l-1]} + t_{P_f[l-1],P_f[l],f} \le Y_{P_f[l]} \qquad l = 1,2,...,n_f,\ f = 1,2,...,\| J \|$$

6). Capacity constraints

$$\sum_{\substack{l=1,2,...,n \\ P_f[l] \in S+}} Z_{m(P_f[l]),k,f,\bullet} - \sum_{\substack{l=1,2,...,n \\ P_f[l] \in S-}} Z_{m(P_f[l]),k,f,\bullet} \le Q_{fk} \qquad 1 < n \le n_f$$

$$f=1,2,...,||J||,\ k=1,2,...,K$$

7). Time window constraints on aircraft servicing time

$$a_f + t_{D_f,P_f[1],f} \le Y_{P_f[1]} \qquad f = 1,2,...,\| J \|$$

$$Y_{P_f[n_f]} + t_{P_f[n_f],D_f,f} \le b_f \qquad f = 1,2,...,\| J \|$$

8). Others

$$Z_{mkfj} \le q_{mk} \quad m=1,2,...M,\ k=1,2,...,K,\ f=1,2,..,\| J \|$$
$$Z_{mkfj} \le Q_{fk} \quad m=1,2,...M,\ k=1,2,...,K,\ f=1,2,..,\| J \|$$
$$X_{ijf}=1 \Rightarrow \exists n,\ 0 \le n \le n_f - 1,\ P_f[n]=i,\ P_f[n+1]=j$$
$$\forall i, j \in S^+ \cup S^- \cup \{D_f \mid \forall f\},\quad f=1,2,...,\| J \|$$

3. Literature review

Our aircraft operations planning problem is a general case of the known vehicle routing problem with time windows (VRPTW).

The VRPTW has been studied extensively during the past three decades. Most of the studies propose either exact or heuristic algorithms (see Bodin, et al (1983), Ball, et al (1995), and Laporte and Norbert (1987)). A typical VRPTW involves the design of a set of minimum cost vehicle routes for a fleet of (typically identical) vehicles, each has a fixed capacity (with respect to a single cargo type), which serves a set of customers with known demands. Each customer has one or more time windows during which service must start, and must be serviced exactly once. All the customers must be assigned to vehicles such that the total demand on any route does not exceed the given capacity. Solving VRPTW optimally is a difficult task. Not only is the VRPTW NP-hard, but even finding a feasible solution when the fleet size is fixed is itself an NP-complete problem (Savelsbergh, 1984).

Assuming a single depot, identical vehicles, and each vehicle can service only one customer at a time, Desrosiers, Soumis and Desrochers (1984) proposed branch-and-bound approaches based on column generation. The columns are generated by a shortest path algorithm with time windows on nodes. A further improved result for this type of approaches is given by Gelinas, et al (1995). Kohl and Madsen (1995) used a Lagrangean relaxation approach given that the objective is to minimize the total distance traveled.

The master problem consists of finding the optimal Lagrangean multipliers, and the subproblem is a time and capacity constrained shortest path problem for each vehicle. An iteration scheme is then proposed to improve the convergence of the subgradient method. Desaulniers, Lavigne and Soumis (1996) also considered this problem assuming that the vehicle waiting time will also cost. They formulated the problem as an integer, nonlinear multi-commodity network flow problem, and applied the column generation techniques embedded in a branch-and-bound algorithm. Both optimal and heuristic versions of the proposed approach were discussed. Dumas, Desrosiers and Soumis (1991) developed an optimization algorithm for VRPTW based on a Danntzig-Wolfe decomposition embedded into a branch-and-bound process. The approach has been successfully used in solving two small real world problems (with number of requests equal to 19 and 30, respectively). However, when the problem size becomes large (e.g., on average 5 requests or more per vehicle), the approach becomes ineffective.

Relaxing the time window constraints, Laporte, Desrochers and Nobert (1984) proposed two exact algorithms for distance-constrained vehicle routing problem. One is based on Gomory cutting plans and the other is based on branch-and-bound. Li, Simchi-Levi and Desrochers (1992) also studied this distance-constrained problem (with time windows relaxed), and performed a thorough analysis on the relationship between minimizing the total distance traveled by vehicles and minimizing the number of vehicles used.

Due to the complexity of VRPTW, most results either published in the literature or used in practice are based on heuristic or local optimization (see Solomon, 1986, Potvin, et al, 1996, Golden, Laporte and Taillard, 1997, etc.). Most heuristic algorithms developed for VRPTW are based on the idea of "cluster first and route second", originally proposed by Bodin and Sexton (1986). With this idea, the given set of customer requests is first partitioned into clusters, and then a single vehicle dial-and-ride problem is solved for each cluster. To name few such heuristics, Roy, et al (1984a, b) proposed a heuristic parallel insertion procedure which simultaneously constructs routes for all vehicles

starting at the beginning of the day by using proximity criteria. The similar approach for minimizing the weighted combination of customer dis-utility and system cost was employed by Jaw, Odoni, Psaraftis and Wilson (1986). Solomon, Baker and Schaffer (1988) accelerated the search speed of branch exchange procedures for route improvement heuristics by eliminating unnecessary checks. Dell'Amico, Fischetti and Toth(1993) analyzed the structural properties of the multiple deport vehicle routing problem with a given set of transportation operations, each has a fixed starting time and ending time. They developed a polynomial heuristic that aiming at minimizing fleet size and overall operational cost. The resulting algorithm always guarantees a minimum fleet size. Potvin, et al (1996) introduced a local optimization based on tabu search that maintains the feasibility of the solution all the time. More recently, Golden, Laporte and Taillard (1997) proposed an adaptive memory heuristic for this type of vehicle routing problem subject to the multiple use of a vehicle and an equal vehicle mileage objective.

There were two studies in the literature that deal with problems very close to ours. One is by Rappoport, et al (1992). To simplify the problem, they partitioned the customer requests before the routes and schedules for vehicles are tested for the feasibility. The partition is done based on best fit, and the routes are constructed based on the shortest path method. The resulting procedure may have to iterate between two phases if the initial schedules are not feasible. Even that, the final schedules may still not feasible due to the complexity of the problem. The other study is by Hooker and Natraj (1995). They proposed a heuristic for airlift operations based on tabu search, and tested the heuristic based on randomly generated problems.

Generally speaking, the vehicle routing problems studied in the literature are more or less a special case of the aircraft operations planning problem that we are considering in this study. First, we have to deal with multiple cargo types on a single aircraft, and different aircraft capacities respect to different cargo types. Second, we have to consider all possible alternatives of transportation, including multiple aircraft to serve a single

customer, or a single aircraft to serve multiple customers at the same time. Third, we have to make sure the aircraft maximum flying hours as a hard constraint.

4. A column generation based approach

In this section, we present a column generation model of the problem. To start, we use *N* (or n) to denote the total number of tasks, where each task contains cargo of only a single type. In reality, we can always divide a given move, based on cargo types, into several tasks, with the same origin, destination, release time and due time. We will use J to denote the set of aircraft, where each aircraft has its own home base, its own maximum capacity with respect to each cargo type, and its own maximum service time (including the flying time, ground time, and waiting time) specified by $[a_j, b_j]$. In addition, we define

a). task i as a seven-tuple:$\{ \pi_i, Q_i, r_i, d_i, o_i, g_i, c_i \}$, i=1,2,...,N

where: π_i is the cargo type of task i;

Q_i is the load of task i; // each task has only a single type cargo

r_i is the release time of task i;

d_i is the due date time of task i;

o_i is the origin location of task i;

g_i is the destination location of task i;

c_i is the penalty for each unit of Q_i uncompleted by its due date;

b). aircraft j as a four-tuple: $\{ a_j , b_j, v(j), C_{jf} \}$, j=1,2,...,||J||

where: a_j stands for the release time of aircraft j from its home base;

b_j stands for the latest returning time of aircraft j to its home base;

v(j) stands for the home base location of aircraft j;

C_{jf} stands for the maximum capacity for cargo type f that aircraft j can carry;

4.1. The Master Problem

Let B_j denote the set of all feasible routes of aircraft j, and

$$\Omega=\{B_1, B_2, \ldots, B_{\|J\|}\}.$$

Let $C^T=[c_1, c_2, \ldots, c_N]$ be the cost vector, and let $U_\varphi=\{U_{1,\varphi}, U_{2,\varphi}, \ldots, U_{N,\varphi}\}$ be the set of unfinished loads under a particular set of routes φ, where

$$\varphi=\{s_1, s_2, \ldots, s_{\|J\|}\} \text{ and } s_j \in B_j$$

Let $\{q^j_{1,s}, \ldots, q^j_{n,s}\}$ be a load vector representing the amount of cargo from each task serviced by aircraft j using route $s \in B_j$. Then, the objective function of our aircraft planning problem can be defined as:

$$\text{Min } U_\varphi C^T = \sum_{i=1}^{N} (Q_i - \sum_{j=1}^{\|J\|} q^j_{i,s}) \cdot c_i$$

$$= \sum_{i=1}^{N} Q_i \cdot c_i - \sum_{i=1}^{N} (\sum_{j=1}^{\|J\|} q^j_{i,s} \cdot c_i)$$

$$= \sum_{i=1}^{N} Q_i \cdot c_i - \sum_{j=1}^{\|J\|} (\sum_{i=1}^{N} q^j_{i,s} \cdot c_i)$$

$$= \sum_{i=1}^{N} Q_i \cdot c_i - \sum_{j=1}^{\|J\|} f^s_j$$

where $f^j_s = \sum_{i=1}^{N} q^j_{i,s} \cdot c_i$ represents the penalty reduced by assigning aircraft j to route s $\in$ Bj. Note that our objective function is equivalent to Max $\sum_{j=1}^{\|J\|} f^j_s$.

Let y^j_s be a binary variable; $y^j_s=1$ if route $s \in B_j$ is used, and $y^j_s=0$ otherwise. Then, our problem can be formulated as the following master problem:

$$\mathbf{P} \qquad \text{Max} \sum_{j=1}^{M} \sum_{s \in Bj} f_s^j \, y_s^j \tag{4-0}$$

Subject to

$$\sum_{j=1}^{\|J\|} \sum_{s \in Bj} q_{i,s}^j \, y_s^j \le Q_i \quad , \quad \text{for} \quad i = 1, \ldots, N; \tag{4-1}$$

$$\sum_{s \in Bj} y_s^j \le 1, \qquad \text{for } j = 1, \ldots, \|J\|; \tag{4-2}$$

$$y_s^j \in \{ 0, 1), \ \forall \ s \in Bj \ , \text{for } j = 1, \ldots, \|J\|; \tag{4-3}$$

Constraints (4-1) ensure that the total amount of cargo from move i do not exceed Q_i, and constraints (4-2) ensure that each aircraft utilizes at most one route.

By relaxing the integer constraints (4-3), we get the linear relaxation of **P**, $\mathbf{P_1}$. Each column in $\mathbf{P_1}$ represents a feasible route of an aircraft. Since the number of routes in set Ω is very large, it is impractical to explicitly list all the columns when solving $\mathbf{P_1}$. Therefore, we shall use column generation approach to generate necessary columns to be further examined during the process of finding the optimal solution to $\mathbf{P_1}$.

Our column generation procedure consists of three main steps:

cg1. Solving a restricted master problem of $\mathbf{P_1}$, call it $\mathbf{P_2}$ which contains a restricted number of columns from (4-1) and (4-2);

cg2. Using the dual variable values of $\mathbf{P_2}$ to solve subproblems (see Section 4.2) to find the column with the maximum marginal cost.

cg3. If a column with positive marginal cost is found, add this new column to $\mathbf{P_2}$ and return to cg1 for the next iteration. Otherwise, the current solution is optimal to $\mathbf{P_1}$. If this solution is an integer, then it is also optimal to **P**. Otherwise, a branch-and-bound process must be explored to get the final integer solution to **P**.

4.2. Formulation of subproblems and subproblem solutions

The solution process by column generation is an iterative process. In each iteration, we solve the restricted master problem $\mathbf{P_2}$ (using the simplex method) to get the values of dual variables, π_i , i=1, ...,N, and α_j , j=1, ..., ||J||, associated with constraints (4-1) and (4-2) respectively. Given the value of dual variables, the marginal cost δ_s^j of the column corresponding to a route s ∈ Bj can be expressed as

$$\delta_s^j = f_s^j - \sum_{i=1}^{N} (q_{i,s}^j \cdot \pi_i) - \alpha_j .$$

Since $f_s^j = \sum_{i=1}^{N} q_{i,s}^j \cdot ci$, we have

$$\delta_s^j = \sum_{i=1}^{N} (q_{i,s}^j \cdot ci) - \sum_{i=1}^{N} (q_{i,s}^j \cdot \pi_i) - \alpha_j$$

$$= \sum_{i=1}^{N} (ci - \pi_i) q_{i,s}^j - \alpha_j$$

Each time after a restricted master problem is solved, we need to solve ||J|| subproblems, one for each aircraft, in order to find the column with the maximum marginal cost. The j-th subproblem is to find a feasible route s ∈ Bj for aircraft j such that its marginal cost is maximized, i.e.,

$$\text{Max } \delta_s^j = \sum_{i=1}^{N} \overline{cij}\, q_{ij}^s$$

where $\overline{cij} = ci - \pi_i$.

To formulate the j-th subproblem, j=1, 2, …, ||J||, let the original location of task i be node i, and the destination location of task i be node n+i. Task i demands that Q_i units be transported from node i to node n+i. Associate with aircraft j, we also add node 2n+j, and node 2n+||J||+j, representing the departure base, and the returning base of aircraft j, respectively. Let θ^j be the set of cargo types that aircraft j can load. Let w_{kf}^j be a

binary constant so that w_{kf}^{j} =1 if the cargo type of task k is included in θ^{j}, and w_{kf}^{j} =0 otherwise.

Now, let **V**= { 1, 2, ..., n, n+1, ..., 2n, 2n+j, 2n+‖J‖+j } be the node set of the network. Let L_1 ={1, 2, ..., n}, L_2 = {n+1, n+2, ..., 2n}, and let L = $L_1 \cup L_2$ be the set of nodes other than the base locations. For each distinct pair of i and k in set **V**, let t_{ik} stand for the traveling time and z_i for the service (ground) time at node i. We then define the following variables:

(1) Binary flow variables X_{ik}^{j} for all i, k ∈ V, i ≠ k.;

X_{ik}^{j} = 1 if aircraft j travels from node i to node k, and

X_{ik}^{j} = 0 otherwise;

(2) Continuous time variables Y_i^j, for all i ∈ L, at which the service at node i starts. Let Y_{2n+i}^{j} , $Y_{2n+\|J\|+i}^{j}$ stand for the time at which aircraft j leaves departure base and arrives at the returning base, respectively;

(3) Continuous load variables H_{if}^{j}, i ∈ L, the total quantity (load) of type f cargo on aircraft j as the aircraft leaves node i, and q_i^j the load from node i (i.e., task i) assigned to aircraft j.

The mathematical formulation for the j-th subproblem can now be given as

SP(j) Max $\delta^j = \sum_{i=1}^{N} \overline{cij}\, q_i^j$

subject to

$$\sum_{k \in V} X_{ik}^j \leq 1, \quad i \in L_1 \tag{4-4}$$

$$\sum_{k \in V} X_{ik}^j - \sum_{k \in V} X_{ki}^j = 0, \qquad \text{for } i \in L \tag{4-5}$$

$$\sum_{k \in L1} X_{2n+j,k}^j \leq 1 \tag{4-6}$$

$$\sum_{k \in L1} X_{2n+j,k}^j - \sum_{k \in L2} X_{k,2n+\|J\|+j}^j = 0, \tag{4-7}$$

$$\sum_{k \in V} X_{ik}^j - \sum_{k \in V} X_{k,n+i}^j = 0, \qquad \text{for } i \in L_1 \tag{4-8}$$

$$Y_i^j + t_{i,n+i} + z_i \leq Y_{n+i}^j, \quad \text{for } i \in L_1 \tag{4-9}$$

$$X_{ik}^j = 1 \Rightarrow Y_i^j + t_{ik} + z_i \leq Y_k^j, \quad \text{for } i, k \in L \tag{4-10}$$

$$X_{2n+j,k}^j = 1 \Rightarrow Y_{2n+j}^j + t_{2n+j,k} \leq Y_k^j, \quad \text{for } k \in L_1 \tag{4-11}$$

$$X_{i,2n+\|J\|+j}^j = 1 \Rightarrow Y_i^j + t_{i,2n+\|J\|+j} + z_i \leq Y_{2n+\|J\|+j}^j, \quad \text{for } i \in L_2 \tag{4-12}$$

$$Y_i^j \geq r_i, \quad Y_{n+i}^j \leq d_i \qquad \text{for } i \in L_1 \tag{4-13}$$

$$Y_{2n+j}^j \geq a_j, \quad Y_{2n+\|J\|+j}^j \leq b_j \tag{4-14}$$

$$X_{ik}^j = 1 \Rightarrow H_{if}^j + w_{kf}^j . q_k^j = H_{kf}^j \quad i \in L, k \in L_1, f \in \theta^j \tag{4-15}$$

$$X_{ik}^j = 1 \Rightarrow H_{if}^j - w_{kf}^j . q_{k-n}^j = H_{kf}^j \quad i \in L, k \in L_2, f \in \theta^j \tag{4-16}$$

$$X_{2n+j,k}^j = 1 \Rightarrow H_{2n+j,f}^j + w_{kf}^j . q_k^j = H_{kf}^j \quad k \in L_1, f \in \theta^j \tag{4-17}$$

$$H_{2n+j,f}^j = 0, \qquad f \in \theta^j \tag{4-18}$$

$$0 \leq q_i^j \leq Q_i \qquad i \in L_1 \tag{4-19}$$

$$H_{if}^j \leq C_{jf} \qquad f \in \theta^j, i \in L \tag{4-20}$$

$$X_{ik}^j \text{ binary}, \qquad i, k \in V \tag{4-21}$$

The objective function seeks to maximize the load assigned to aircraft j, i.e., to minimize the penalty. Constraints (4-4) ensure that each task can be picked at most once (assuming that the load of each task is no more than the maximum capacity of the aircraft for the respective cargo type). Constraints (4-5) are the flow conservation constraints. Constraints (4-6) ensure the aircraft j to be used at most once. Constraints (4-7) ensure that if the aircraft is used, it must finally return to its destination depot. Constraints (4-8) ensure that once a task is loaded, it must be unloaded to its designated destination, i.e., the same aircraft must visit both nodes i and 2n+i. Constraints (4-9) enforce node i to be visited before node 2n+i is visited. Constraints (4-10)-(4-12) describe the relationship between the flow variables and the time variables, and constraints (4-13) and (4-14) are the time window constraints. Constraints (4-15)-(4-17) describe the relationship between the flow variables and the aircraft's load for each cargo type on the route. Finally, Constraints (4-18)-(4-20) are the capacity constraints.

When the subproblem does not generate any more columns with positive marginal cost, the simplex algorithm provides the optimal solutions to $\mathbf{P_1}$. As we have discussed in section 4.1, if this solution is integer, then it is also optimal to **P**. Otherwise, a branch-and-bound process must be explored to get the final integer solution to **P**.

4.3. Discussion

While the restricted master problem $\mathbf{P_2}$ is a linear program, its subproblems are NP-hard (i.e., the vehicle routing problems with time window and other side constraints). This makes the approach of column generation relatively inefficient for our problem. Nevertheless, the model presented in this section can always be used for handling

problems with relatively smaller sizes. When the problem is large, the approach that we shall discuss in Section 7 can be used to generate feasible routes in a very efficient way.

5. An integer program for assigning moves to aircraft

In this section, we propose an integer programming model that assigns moves of different types to aircraft to maximize the utilization of the aircraft capacity. This model assumes the following problem. We are given a set of moves with different types (some moves may require the services by multiple heterogeneous aircraft). We are also given a set of aircraft of different types, in terms of their own maximum capacity for each cargo type. The problem is to determine the optimal number of aircraft of each type to be assigned to each move so that the total amount of unutilized aircraft capacity is minimized.

This model does not consider the traveling and time window issues. However, its solution does provide helpful decision information about how the aircraft capacity can be best utilized and how many each type of aircraft should be allocated to each move.

5.1. The integer programming formulation

To start, let's introduce the following notation. Let

π = The total number of aircraft types, j=1, 2, …, π,

M = The set of moves to be delivered;

K = The total number of cargo types specified by the moves in M;

$\theta(m)$ = The set of cargo types of move m, $1 \le \theta(m) \le K$;

V_{jm} = The number of type j aircraft assigned to move m;

q_{mk} = Load requirement by cargo type k of move m;

Q_{jk} = Aircraft type j's maximum capacity for cargo type k;

$$\min \quad \sum_{m=1}^{\|M\|} \sum_{k=1}^{K} [(\sum_{j=1}^{\Pi} V_{jm} \cdot Q_{jk}) - q_{mk}] \qquad (5\text{-}1)$$

s.t.

$$\sum_{j=1,\ldots,\pi} V_{jm} \cdot Q_{jk} \geq q_{mk}, \quad k=1,2,\ldots,K,\ m=1,2,\ldots,\|M\| \qquad (5\text{-}2)$$

$$V_{jm} \quad \text{integers, } j=1,2,\ldots,\pi,\ m=1,2,\ldots,\|M\| \qquad (5\text{-}3)$$

The objective (5-1) is to minimize the total amount of unutilized aircraft capacity assigned to the moves, where quantity

$$\sum_{j=1}^{\Pi} V_{jm} \cdot Q_{jk}$$

stands for the total capacity for cargo type k, contributed by all the aircraft of different types allocated to move m. Constraints (5-2) ensures that each of the moves in set M must be given enough aircraft capacity.

This integer programming model is totally decomposable, and can be solved independently for each move. In other words, each move defines a block in the constraint set of the model. The size of a block, and thus the computational effort, depends only on the total cargo types (which is typically less then five) and the total number of aircraft type (which is also a very limited number in reality). While the number of moves in practice could go to a very large number, it does not affect the model complexity at all. Furthermore, if we are only interested in minimizing the waste of the *effective capacity*[1] of aircraft assigned to a move, then (5-1) can be reduced to

$$\min \quad \sum_{m=1}^{\|M\|} \sum_{j=1}^{\pi} V_{jm} \cdot \sum_{k=1, k\in\theta(m)}^{K} Q_{jk} \qquad (5\text{-}4)$$

[1] For a given assignment of aircraft j to move m, the effective capacity of aircraft j equals to

$$\sum_{k=1\ldots,K,\, k\in\theta(m)} Q_{jk}$$

Any commercial mathematical programming package can be easily applied to solve this model. The following example is solved by using the linear programming tool, LINDO.

5.2 A numerical example

We are given the following set of moves and aircraft.

Moves and load requirements

Moves	**Passenger** (cargo type 1)	**Bulk** (cargo type 2)	**Oversize** (cargo type 3)	**Outsize** (cargo type 4)	$\theta(m)$
m=1	948				{1}
m=2	401	107			{1, 2}
m=3	231	498	2166		{1, 2, 3}
m=4	985	1750	9890	690	{1,2,3,4}
m=5		628	2528	164	{2, 3, 4}

Maximum capacity for cargo type

Aircraft	Passenger	Bulk	Oversize	Outsize
Type 1: B747	401			
Type 2: C141	22	30	30	
Type 3: C5	73	83	72	78
Type 4: DC10		74	37	
Type 5: DC8		52		

Assuming objective function (5-4) is used, the resulting integer programming model has the following format.

Min $401V_{11}+22V_{21}+73V_{31}+401V_{12}+52V_{22}+156V_{32}+74V_{42}+52V_{52}+401V_{13}+82V_{23}+228V_{33}$
$+111V_{43}+52V_{53}+401V_{14}+82V_{24}+306V_{34}+111V_{44}+52V_{54}+60V_{25}+233V_{35}+111V_{45}+52V_{55}$

S.t

$401\ V_{11}+22\ V_{21}+73\ V_{31} \geq 948$

$401\ V_{12}+22\ V_{22}+73\ V_{32} \geq 401$

$30\ V_{22}+83\ V_{32}+74\ V_{42}+52\ V_{52} \geq 107$

$401\ V_{13}+22\ V_{23}+73\ V_{33} \geq 231$

$30\ V_{23}+83\ V_{33}+74\ V_{43}+52\ V_{53} \geq 498$

$30\ V_{23}+72\ V_{33}+37\ V_{43} \geq 2166$

$401\ V_{14}+22\ V_{24}+73\ V_{34} \geq 985$

$30\ V_{24}+83\ V_{34}+74\ V_{44}+52\ V_{54} \geq 1750$

$30\ V_{24}+72\ V_{34}+37\ V_{44} \geq 9890$

$78\ V_{34} \geq 690$

$30\ V_{25}+83\ V_{35}+74\ V_{45}+52\ V_{55} \geq 628$

$30\ V_{25}+72\ V_{35}+37\ V_{45} \geq 2528$

$78\ V_{35} \geq 164$

V_{jm} integers, j=1, 2,…,5, m=1, 2,…,5.

The resulting IP solution gives:

The need of aircraft type 1: $V_{11}=2$, $V_{12}=1$

The need of aircraft type 2: $V_{23}=71$, $V_{24}=309$, $V_{25}=76$

The need of aircraft type 3: $V_{31}=2$, $V_{34}=9$, $V_{35}=3$,

The need of aircraft type 4: $V_{42}=1$, $V_{43}=1$, $V_{45}=1$

The need of aircraft type 5: $V_{52}=1$

This solution implies:

Aircraft	Best Usage	Capability	Primary Responsibility
B747	moves 1,2	1, [2], [3], [4]*	
C141	moves 3,4,5	1, 2, 3, [4], [5]	
C5	moves 1,4,5	1, 2, 3, 4, 5	4, 5
DC10	moves 2,3,5	[2], [3], [4], [5]	
DC8	Move 2	[2], [3], [4], [5]	

[m]* For given move m, the respective aircraft type can service only certain cargo types in $\theta(m)$, and therefore can not be used alone for move m.

6. Minimizing the fleet size with fixed aircraft release time at destinations

In this section, we present a simple and fast algorithm based on the maximum flow theory (Ahuja, Magnanti and Orlin, 1993). This algorithm determines the minimum fleet size required to deliver a given set of moves within their specified time windows, and is based on the following model.

We are given a set of moves, M, a single home base s for the aircraft to depart, and a single base t for the aircraft to return, which may be different from s. Each move i, $i \in M$, has an origin o_i, a destination g_i, a release time r_i, and a due time d_i with

$$d_i \geq max\{t_{s,i}, r_i\} + \tau_i$$

where τ_i denotes the required time to delivery move i, and $t_{s,i}$ denotes the traveling time from s to o_i. There are three additional assumptions for this model. First, it is assumed that each aircraft service only one move at a time (i.e., an aircraft may service other moves after the current move has been completed). Second, in the case that an assigned aircraft arrived at the destination of a move earlier than the due time, the aircraft will remain there until the due time. After that, the aircraft may travel to the origin of some other move for a different assignment. Third, an aircraft can be used to service any move in M, which is equivalently to say that all the aircraft are identical. The problem is to find the routes for aircraft so that the total number of aircraft needed to service the given set of moves is minimized while the time windows for all the moves are satisfied.

The resulting algorithm is a strongly polynomial time procedure based on maximum flow theory. While the underlying model requires additional assumptions, the methodology can be used to find a quick estimate for the fleet size needed to transport moves within their time windows. It can also be used to estimate an upper bound on the fleet size when aircraft are allowed to leave early before the due time. Such estimation could be very helpful for the capacity planning purposes.

6.1. Network representation for the problem

Given M, we can construct a network G(V,A), where V(G))=$\{s, t, o_i, g_i \mid \forall i \in M\}$, $\|V(G)\|=2(\|M\|+1)$, stands for the set of nodes, and A(G) stands for the set of arcs on G. For each $o_i \in G$, we add an arc from s to o_i with $l_{s,i}= 0$, $u_{s,i}=1$, and for each $g_i \in G$, we add an arc from g_i to t with $l_{i,t}= 0$, $u_{i,t}=1$, where $l_{i,j}$, and $u_{i,j}$, stands for the lower, and upper, bound on the flow through arc (i, j), respectively. For each move i, $i \in M$, add an arc from o_i to g_i with $l_i = u_i =1$. In addition, we add an arc from g_i to o_j iff

$$d_i + t_{i,j} + \tau_j \leq d_j \qquad (6\text{-}1)$$

where $t_{i,j}$ stands for the required time to travel from the destination of move i to the origin of move j. Condition (6-1) ensures a feasible assignment for an aircraft to service move j after it has completed move i, $i, j \in M$. The total number of arcs on G(V,A) is bounded from above by $\|A(G)\|=O(\|M\|^2)$.

The problem is to find a minimum flow from s to t with all the flow constraints satisfied. Based on the definition of bounds on flows, we see that a feasible flow satisfying the given bounds ensures all the moves to be serviced.

6.2. A strongly polynomial algorithm for solving the problem

Given the model introduced at the beginning of section 6 and a network G(V,A) with lower and upper bounds on each arc, the following algorithm determines the minimum fleet size required to deliver all the moves within their respective time windows.

Step 1. Find a feasible flow v_1 from s to t on G(V,A) that satisfies all the lower and upper bounds. This can be done by simply using the trivial feasible solution that sends one unit of flow through each path s-o_i-g_i-t, for all $i \in M$, which results in $v_1=\|M\|$. Let $x_{i,j}$ be the resulting solution, where $x_{i,j}$ stands for the amount of flow from node i to node j on G.

Step 2. Construct the *residual network* $\boldsymbol{R}$ based on the given initial feasible flow $x_{i,j}$ (see the definition of residual network (Ahuja, Magnanti and Orlin (1993), pp192-193);

Step 3. Find the maximum flow v_2 from $\boldsymbol{t}$ back to $\boldsymbol{s}$ based on residual network $\boldsymbol{R}$. Ley $y_{i,j}$ be the flow from i to j determined by solving the maximum flow problem on $\boldsymbol{R}$.

Step 4. The minimum fleet size is given by $Z^* = v_1 - v_2$. The resulting route for each aircraft can be determined as follows: for each arc (g_i, o_j) on the residual network with $y_{i,j} > 0$, add an one unit flow on the original network from g_i to o_j, remove the flow from $\boldsymbol{s}$ to o_j, and remove the flow from g_i to $\boldsymbol{t}$ (i.e, let $x_{s,oj}=0$, $x_{gi,t}=0$). Each unit flow from $\boldsymbol{s}$ to $\boldsymbol{t}$ on the revised original network specifies the route for each aircraft. A total Z^* unit flows from $\boldsymbol{s}$ to $\boldsymbol{t}$ are therefore generated as the algorithm terminates.

The time effort required by this algorithm to find the optimal solution to the fleet size planning problem defined at the beginning of this section is $O(\|M^2\|)$, which is mainly the time effort required to solve the maximum flow problem (Ahuja, Magnanti and Orlin (1993)).

Example 6-1. Given a set of eight (8) moves and the resulting network, where we have an arc from g_i to o_j if it is possible for an aircraft to continue to service move j after it completes move i, that is, if $d_i + t_{gi,oj} + \tau_j \leq d_j$ is satisfied.

The initial feasible flow from home base $\boldsymbol{s}$ to returning base $\boldsymbol{t}$, that satisfies the lower and upper bounds, is $v_1=8$ (see Figure 6-1). The maximum flow from returning base $\boldsymbol{t}$ back to home base $\boldsymbol{s}$ based on the residual network is $v_2=5$ (see Figure 6-2). This results in a minimum fleet size $Z^*=v_1-v_2=3$. Applying Step 4 of the algorithm to determine the optimal tours for the $Z^*=3$ aircraft, we obtain the following:

1). Remove arcs (s, o_2), (s, o_5), (s, o_6), (g_3, t), (g_2, t) and (g_5, t) on the original network to form tour

s—o_3—g_3—o_2—g_2—o_5—g_5—o_6—g_6—t

for the first assigned aircraft departing from $\boldsymbol{s}$;

2). Remove arcs (s, o_7), (s, o_8), (g_1, **t**) and (g_7, t) on the original network to form tour

$$s—o_1—g_1—o_7—g_7—o_8—g_8—t$$

for the second assigned aircraft departing from ***s***; and

3). (We now have only one move, move 4, left) The tour for the third aircraft departing from ***s*** is $s—o_4—g_4—t$.

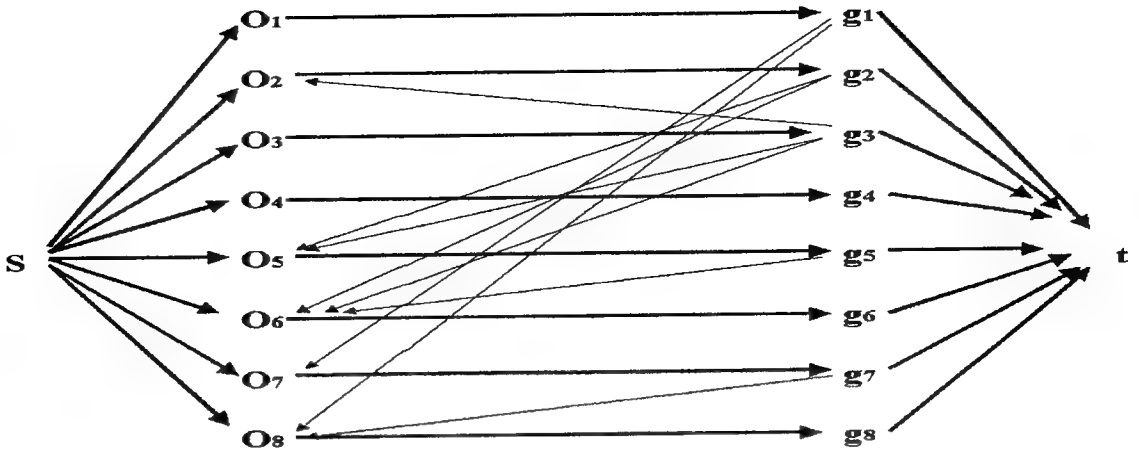


Figure 6-1. Initial flow of example 6-1.

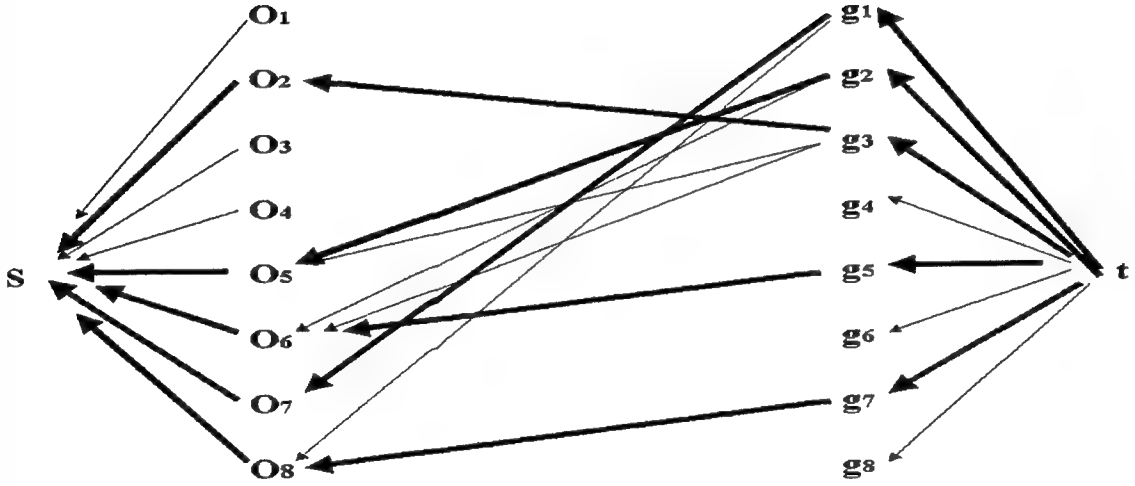


Figure 6-2. Maximum flow on residual network of example 6-1.

7. The Greedy-Tour heuristic for the general operations planning problem

7.1 Assumptions for the model

We are given a set of aircraft, $\boldsymbol{J}$, each is originally located at its home base (i.e., we allow multiple depots in this discussion). Aircraft can be different in terms of their capacity for different cargo types (e.g., DC10 has a capacity for 74 units of bulk cargo and 37 units of oversize cargo, while B747 has a capacity only for 401 passengers). It is possible to have more than one aircraft from the same home base. In addition to the heterogeneity in capacities, aircraft of different types may also have different speed for traveling and different requirement for ground time. There is a maximum flying hour limit, L, for each aircraft. Let Π be the total number of aircraft types involved.

We are also given a set of moves, $\boldsymbol{M}$. Each move i is defined by $[o_i, g_i, r_i, d_i, \tau_i]$, i=1, 2, ...,$\|\boldsymbol{M}\|$, where parameters o_i, g_i, r_i, d_i, and τ_i represents the origin, destination, release time, due time, and the traveling time for delivering move i, respectively. A move may have cargo of different types and its own requirements on aircraft capacities.

The problem can be either to determine a set of alternative feasible routes, or a single "most-rewarded" route, for each aircraft in **J** so that all the constraints are satisfied.

7.2. General description of the heuristic

We propose a Greedy-Tour heuristic for solving the problem defined in section 7.1. This heuristic can be used either to find a set of alternative feasible routes, or to determine the "most-rewarded" route for each given aircraft in $\boldsymbol{J}$. Our discussion in this section will be based on the objective to determine the most rewarded route for each aircraft in $\boldsymbol{J}$.

A route is called a "most-rewarded" route if it services as many moves as possible while satisfying the time windows (i.e., $[r_i, d_i]$), the maximum flying hours (i.e., L), and the capacity constraints. Let J_j be a subset of aircraft that are of type j, $1 \leq j \leq \Pi$, then

$$J = J_1 \cup J_2 \cup ... \cup J_\Pi .$$

This GT heuristic algorithm consists of Π main loops, where the j-th loop (defined by Steps 1, 2, and 3 below), j=1, 2,..., Π, constructs a total of $\| J_j \|$ most-rewarded routes for the aircraft in set J_j. Steps 1, 2, and 3 in each loop are explained as follows:

Step 1. For the given aircraft type j, $1 \leq j \leq \Pi$, identify the subset of moves (from M which contains remaining moves) that can be serviced by aircraft in set J_j. Restructure the moves in this subset so that each *restructured move* has a capacity no more than, and no less than 50% of, the aircraft capacity so that it makes sense to let an aircraft of type j to service only one *restructured move* at a time. Note that such a restructured move could either be the one decomposed from a very large move, or be the one from combining several small moves. Forming a combined move requires determining the *efficient subtour* that connects several small moves. For example, to form a combined move for two small moves, move i and move h, we have two alternative subtours.

$$o_i -> o_h -> g_i -> g_h,$$
$$o_i -> o_h -> g_h -> g_i.$$

The one that completes the subtour earlier is the efficient tour. Since a combined move usually consists of only very few small moves, identifying the efficient subtour is not an issue. We shall use V_j to denote the resulting set of restructured moves.

Step 2. Given V_j and J_j, we construct a directed graph $G(V_j, A_j)$ where V_j is the set of nodes and A_j is the set of arcs. Each node in V_j corresponds to a restructured move and can be serviced by any aircraft in set J_j. Note that we do not have separate nodes for the origin and the destination of a move in this model. For any pair of moves i and h, we draw an arc from node i to node h if and only if

$$r_i + \tau_i + t_{g_i,o_h} + \tau_h \leq d_h \qquad (7\text{-}1)$$

which means that moves i and h can be two consecutive moves assigned to the same aircraft without violating the time window constraints. The reward assigned to arc (i, h) is

$$c_{i,h} = p_h$$

where p_h is a positive constant representing a reward for servicing move h. This means the path with the maximum total reward is the path that services the most moves (or more important moves).

Step 3. Given J_j and $G(V_j, A_j)$, this step will perform a local iteration. Within each iteration, we select the most urgent move (i.e., the move with the earliest due time) from current V_j as the source location on $G(V_j, A_j)$, and call this move i_0. We then determine the shortest path from i_0 to every other node on the graph to form a shortest-path tree. A revised shortest path tree algorithm (to be explained section 7.3) is used to construct this tree (rooted at node i_0). This revised shortest path tree algorithm is able to construct the tree with all the constraints satisfied. During the tree construction process, whenever the length of a branch on the tree is close enough to L (i.e., the maximum flying hours for each aircraft), the branch is banned from further growing. After the tree is completed, the branch/path with the most moves (i.e., the most-rewarded path) is selected. We choose an aircraft in set J_j which has a home base, say location y, nearest to the origin of node i_0, and assign the aircraft to service the moves ordered by the most rewarded path. Appending node y to the beginning, and the end, of the path, a route is formed. We then remove all the moves on this most rewarded path from V_j, and remove the assigned aircraft from J_j. Each iteration forms a route for an aircraft in J_j and reduces the size of V_j and J_j. Step 3 terminates when either V_j or J_j (or both) becomes empty.

Note that if J_j becomes empty prior to V_j, then the remaining moves in V_j will be combined with those left in current set M and be considered in the next main loop.

7.3. The revised shortest path tree algorithm for feasible tours

For a given directed graph $G(V_j, A_j)$ and a specified source/root node i_0, we need a algorithm that constructs the shortest path tree that satisfies all the constraints. In this section, we propose such an algorithm based on the idea of generalized bucket by Desrochers and Soumis (1988).

Desrochers and Soumis (1988) proposed a Generalized Permanent Labeling algorithm for solving the shortest path problem with time window constraints. In their model, each node represents to a unique location to be visited and has a time window $[a_i, b_i]$ so that the visiting time must take place within that window. Each path from the source node to a particular node is assigned with a unique label (T,C) where T represents the arrival time and C represents the cost of the path. Let X_1 and X_2 be two arbitrary paths from the source node to a particular node with associated labels (T_1, C_1) and (T_2, C_2). Then path X_1 dominates path X_2 if and only if $T_1 \le T_2$, $C_1 \le C_2$, and $(T_1, C_1) \ne (T_2, C_2)$. A label at a given mode is *efficient* if and only if no other label at that node dominates it. Their algorithm is based on the concept of a generalized bucket and on a specific order of handling efficient labels. A bucket contains a list of nodes whose label values lie within a specific interval (see Figure 7-1 for an example of bucket). In the simple case, the kth bucket contains all such nodes whose T values fall in the semi-open interval $\min\{a_i \mid \forall i\} + [(k-1)m_d, km_d)$ where

$$m_d = \min\{t_{i,j} \mid \forall (i,j) \in A\}$$

and t_{ij} stands for the distance from node i to node j.

Different from their study, we also need to deal with the situation where each move has a distinct origin and a destination, and where the shortest path tree can not allow the length of any of its branches to be more than the maximum flying hours L, which makes our problem more complicated.

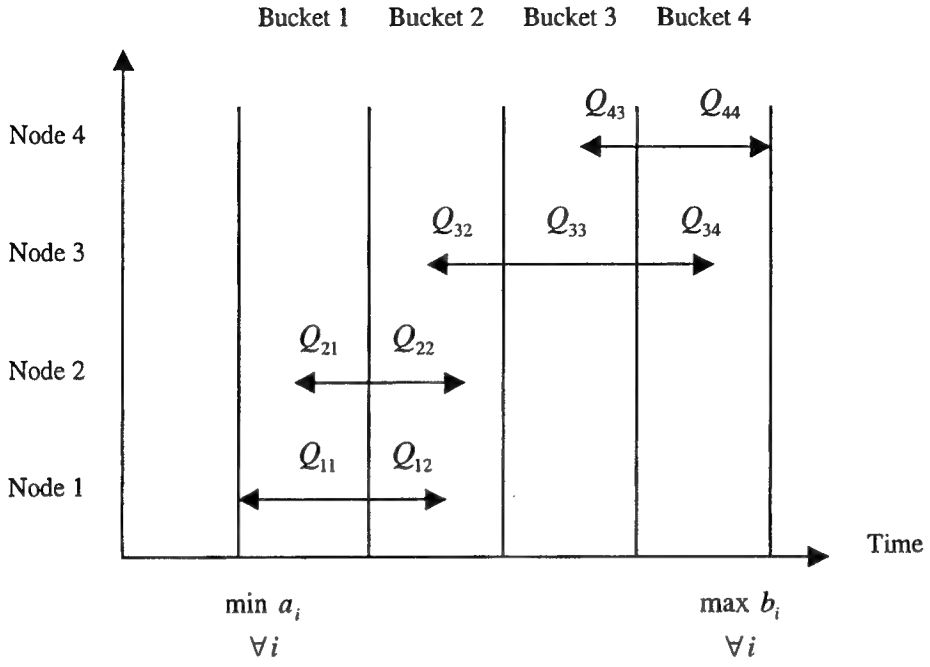


Figure 7-1. An example with four buckets, where Q_{ij} is a subset of efficient labels for node i with respect to bucket j.

Based on the idea of the generalized bucket, we propose a revised shortest path tree (RSPT) algorithm. This RSPT algorithm consists of the following two steps:

Step 1. Divide the entire interval $[\min\{r_i \mid \forall i\}, \max\{d_i \mid \forall i\}]$, where r_i, and d_i, stands for the move release time, and move due time, respectively, into subintervals of width m_d. Parameter m_d is given by

$$m_d = \min\{t_{i,j} \mid \forall (i,j) \in A\} \qquad (7\text{-}2)$$

where $t_{i,j}$ stands for the traveling distance from the destination of move i to the origin of move j. Let K be the total number of subintervals (i.e., the total number of buckets) where the k-th subinterval, $1 \le k \le K$, is defined as

$$\min\{a_i \mid \forall i\} + [(k-1)m_d, km_d).$$

For each node i, if its time window $[r_i, d_i]$ overlaps with the k-th subinterval, then node i has a subset of efficient labels, $Q_{i,k}$, associated to bucket k. Depending on the width of $[r_i, d_i]$, node i may have several subsets of efficient labels. An example with four nodes

and four buckets is given in Figure 7-1, where bucket 1 contains nodes 1 and 2, bucket 2 contains nodes 1, 2 and 3, and both buckets 3, and 4, contain nodes 3 and 4, respectively.

Step 2. This step constructs the shortest path tree. During the construction process, all subsets of efficient labels in the first bucket are processed, and then all subsets of efficient labels in the second bucket are processed, and so on, until all subsets of efficient labels in the last bucket are processed. To process a label, which corresponds to a path from the given source node i_0 to a particular node i in $G(V_j, A_j)$, all the immediate successors of node i are examined. For each such successor, the shortest path, as well as the total reward based on the shortest path, from the source node i_0 through node i to that successor are established. Let node i be the node being processed, let z be one of its immediate successors, then the label of the path *from i_0 to z through i* is given by

$$(T_z, C_z) = (\max(r_z, T_i + t_{g_i,o_z}\} + \tau_z,\ C_i + c_{i,z}) \quad \text{if } \max(r_z, T_i + t_{g_i,o_z}\} + \tau_z \le \min\{d_z, L\} \tag{7-3}$$

where T_z stands for the earliest time to complete move z by using the path *from i_0 to z through i*. Condition (7-3) means that whenever the resulting $T_z > \min\{L, d_z\}$, we stop the branch that tries to further extend from node i to node z. This will ensure the time windows (on moves) and the maximum flying hours to be satisfied. Note that we may still allow another branch that extends from node i to some other node z, if $T_{z'} \le \min\{L, d_{z'}\}$. Repeat this step until all the efficient labels in the K buckets are processed which leads to a feasible shortest path tree subject to all the constraints specified in section 7.1.

The pseudo code of this RSPT algorithm will be given in section 7.4.

7.4. Pseudo-code for the Greedy-Tour (GT) heuristic

This section presents two sets of pseudo code. One is for the revised shortest path tree (RSPT) algorithm (see Table 7-1), which will be used as a subroutine in the Greedy-Tour (GT) heuristic. The other is for the GT heuristic (see Table 7-2). The general descriptions of these algorithms are in sections 7.2 and 7.3 respectively.

Table 7-1. Pseudo code for the revised shortest path tree algorithm

The Revised Shortest Path Tree (RSPT) Algorithm (that builds a set of alternative feasible routes departing from source node i_0 for each aircraft)

{Input: The directed graph containing restructured moves, $G(V_j, A_j)$;
The width of bucket, m_d, as defined by (7-2);
The total number of buckets, K;
The given source node i_0; and
The maximum flying hours, L}

Step 1. *Initialization*

$$Q_{i,k} = \begin{cases} \varnothing & \text{If } 1 < k \le K,\ \forall i \in V_j,\ i \ne i_0 \\ \{(0,\ 0)\} & \text{If } k = 1,\ i = i_0 \end{cases}$$

where $Q_{i,k}$ is the subset of efficient labels of node i associated to bucket k.

Step 2. *Loop to build a feasible shortest path tree rooted at source node i_0*

For bucket k=1, 2, …, K, do
For each node i with a label set $Q_{i,k}$ associated to bucket k, do

Step 2.1. • Get the next label in set $Q_{i,k}$, say label (T_i, C_i);

Step 2.2. • For all successors z of node i, do

If $\max\{r_z, T_i + t_{g_i,o_z}) + \tau_z \le \min\{d_z, L\}$

Then $(T_z, C_z) = (\max\{r_z, T_i + t_{g_i,o_z}) + \tau_z, C_i + c_{i,z})$

Let u be the bucket containing T_z, and let

$Q_{z,u} = EFF(Q_{z,u} \cup (T_z, C_z))$

Step 2.3. • Goto Step 2.1 until all the labels in the current set $Q_{i,k}$ are processed;

Table 7-2. Pseudo code for the Greedy-Tour (GT) algorithm

*The Greedy-Tour (GT) algorithm (that assigns each aircraft with a most-rewarded feasible route, for all the aircraft in set **J**)*

*{Input: The set of aircraft, **J**; The set of moves, **M**;*
*Output: The set of ||**J**|| most-rewarded tours}*

Step 1. For aircraft type j = 1, 2, ..., Π, do
*{Let V_j be the a subset of remaining moves in **M**. Set V_j consists of those moves that can be serviced by aircraft of type j, and let J_j be the set of aircraft of type j}*

Step 1.1. For given aircraft type j, restructure the moves in V_j (see section 7.1) to make the move load matches the capacity of aircraft of type j. Construct network $G(V_j, A_j)$, where each node in V_j stands for a restructured move, and A_j stands for the set of arcs. For any pair of nodes in V_j, say i and h, we add an arc from node i to node h iff

$$r_i + \tau_i + t_{g_i,o_h} + \tau_h \le d_h$$

and let $c_{i,h} = p_h$ where constant $p_h>0$ represents a positive reward for servicing move h.

Step 1.2. While $J_j \neq \varnothing$ and $V_j \neq \varnothing$ do

- Let i_0 be the most urgent move in current V_j so that
$$d_{i_0} = \min\{d_i \mid \forall i \in V_j\};$$
- Call the RSPT subroutine and construct a shortest path tree with root i_0;
- Given a shortest path tree, determine the leave node, say i^*, on the tree that has the maximum reward, i.e., $C_{i^*} = \max\{C_i \mid \forall i \in V_j\}$. Call the path from i_0 to i^* the *most-rewarded path*. Determine the nearest home base, y, to the root i_0, and dispatch an aircraft of type j from home base y to the most rewarded path.
- Remove all the (no home base) nodes on the most rewarded path from V_j and remove the associated aircraft from J_j.

Step 1.3. If $V_j \neq \varnothing$, then $M \Leftarrow M \cup V_j$;

Step 2. Print out the set of $\|J\|$ most-rewarded routes. If the remaining $M \neq \varnothing$, then print out the remaining moves in M that can not be serviced within their respective time windows under the given fleet J.

8. The Sequential assignment method for the minimum-cost operations plan

In this section, we propose a fast and effective computer-based planning method, the sequential assignment method. This proposed planning method has three major advantages. First, it can handle many complicated constraints without the need of formulating them into a formal mathematical program, as long as these constraints can be expressed as logical statements in a computer programming language. Second, the main component of this method is a subroutine for solving the well-known assignment problem, to which many efficient algorithms are available. This makes the planning method easy to implement and can run very fast. Third, since our proposed approach is based on a sequential planning process, it can be used in real time on a continuous basis. That is, it can be used to keep updating/extending operation plans as new moves coming in, without restarting the planning process. This planning method does not require us to restructure the moves. All the raw data can be used as they are.

8.1. Modeling assumptions

Our proposed sequential assignment method is based on the following model. We are given a set of aircraft, $\boldsymbol{J}$, each is originally located at its home base (i.e., we allow multiple depots in this discussion). Aircraft are heterogeneous in the sense that different aircraft may have different traveling speed, different ground time, and different capacity limit for passengers, bulk cargo, oversize cargo and outsize cargo, respectively. Each aircraft may also have its own maximum total flying hours (i.e., the maximum flying hours between aircraft maintenance). We assume that a cost is incurred whenever an aircraft is assigned to service a move.

We are also given a set of moves, $\boldsymbol{M}$. Each move i is defined by $[o_i, g_i, r_i, d_i, \tau_i]$, $i=1, 2, ...,\|\boldsymbol{M}\|$, where parameters o_i, g_i, r_i, d_i, and τ_i represents the origin, destination, release time, due time, and the traveling time for delivering move i, respectively. A move

may have cargo of different types, and different moves may have different requirements on aircraft capacities. For example, one move may have 4000 passengers only, while the other move may have no passengers, but 200 units bulk, 500 units oversize, and 700 units outsize cargo, respectively. Associated with each move, there is also an assigned priority so that moves of higher priority will be considered first.

The problem is to determine a route for each aircraft in J so that the total cost is minimized, while 1) all the constraints (i.e., the time windows of moves, the capacity and the maximum flying hours of aircraft) are satisfied; and 2) moves of higher priorities are being serviced as many as possible if the aircraft resource is insufficient.

8.2. The sequential assignment method for operations planning

Our proposed sequential assignment method refers to an iterative (sequential) planning process. The iteration continues until either all the aircraft are exhausted (i.e., have reached their maximum flying hour limits) or no more remaining moves left.

Each iteration starts with a given set of aircraft, together with their current location and remaining flying hours before the maintenance, and a set of remaining moves. For each aircraft, we compute the cost if this aircraft is assigned to service each of the remaining moves. For each potential assignment of an aircraft to a move, we say the assignment is feasible if

a). The aircraft has the capacity to service (at least part of) the cargo of that move;
b). The aircraft has enough remaining flying hours for delivery the move; and
c). Given the other moves that are already assigned to this aircraft, it is still able to deliver the move within the due time.

For each such a feasible pair of aircraft and move, say move m, we assign the cost as

$$T_{i,j} / p_m$$

where p_m stands for the priority of move *m* (the more critical a move is, the higher priority value it will have), and $T_{i,j}$ stands for the total traveling time from the aircraft's current location (location i) to origin of move *m* plus the total traveling time from origin of move *m* to its destination (location j), including the loading and unloading time. After the costs for all such potential pairs are estimated, we then solve the associated assignment problem to make the most economic assignment of aircraft to remaining moves. Note that in each iteration, if the number of aircraft does not equal to the number of remaining moves, then either some aircraft will not have the assignment or some moves will not be considered in the current iteration. Based on the solution to the assignment problem, moves being selected will be added to the route of the respective aircraft. We then expand the routes of aircraft accordingly and update their status (e.g., the aircraft's new location and the remaining flying hours), and prepare the remaining move for the next iteration. If the capacity of an assigned aircraft is less than the load of a move, the move is not completed. The remaining load of that move will become a new move in the following iteration(s).

A step-by-step description of this sequential assignment method is as follows.

Step 1. Given a set of aircraft ***J*** together with their status and a set of moves ***M***, construct a bipartite graph (see the definition of bipartite graph on page 31 of Ahuja, Magnanti and Orlin (1993)). For each "aircraft", a circle-node is created. For each move, a square-node is created. An arc between a circle-node and a square-node is created if the respective aircraft can be assigned to the corresponding move without violating all constraints. If an arc is created, then a cost is assigned to this arc. If no more arc can be created, then go to Step 4.

Step 2. Solve the associated assignment problem (any available algorithm in the literature for solving the standard assignment problem can be used for this step), such that each circle-node is assigned to a square-node so the total cost is minimized.

Step 3. Update node information (i.e., update the status of aircraft and moves). Based on the solution from Step 2, if circle-node i (aircraft i) is assigned to square-node j (move j), then the information of node i (such as the associated flight starting time, finish time, aircraft location after delivering the newly assigned move, and remaining flying hours) and the information of node j (such as the remaining load with respect to each cargo type) are updated.

Step 4. Print out the assignments obtained so far which defines the routes for aircraft.

The resulting planning process continues until no more arc can be created. That is, it continues until either all the aircraft are exhausted or no more moves are left.

8.3. Specifications on the computer program for the sequential assignment method

We have attached a computer program that implements the proposed sequential assignment method in Appendix (coded in C++). This software requires four input data files. They are

a). Data file <aircraft.data>

It specifies the set of aircraft and their home bases in the following format:

	Aircraft name	home base location
e.g.	B747-P	TAMPA

b). Data file <move.data>

It specifies the set of moves in the following format:

	Move-name	Origin	Destination	Release-time	Due-time	Passenger	Bulk	Oversize	Outsize	Priority
e.g.	T10	BOSTON	TAMPA	12	14	1200	10	25	33	1

c). Data file <aircraft.types>

It specifies the information about aircraft types in the following format:

	Aircraft-name	Passenger	Bulk	Oversize	Outsize	Speed	Load-time	Unload-time	Max-flying-hour
e.g.	B747-P	401	0	0	0	500/hr	27 (minutes)	27 (minutes)	40 (hrs)

where the load/unload time are in minutes, and the maximum flying time is in hours.

d). Data file <location.xy>

It specifies the geographical location information of depots, origins and destinations of moves in the following format:

	home base/origin/destination Location	Longitude	Latitude
e.g.,	BOSTON	71090310	42319644

The output file contains the set of routes, one for each aircraft, generated by the proposed sequential assignment method in the following format, for example

B747-P Base: TAMPA

T5:	TAMPA(1, 0:00)->CHICAGO	CHICAGO(8, 0:00)->HOUSTON
T6:	HOUSTON(8, 1:00)->LOS_ANGELES	LOS_ANGELES(13, 0:00)->TAMPA
T6:	TAMPA(13, 5:00)->LOS_ANGELES	LOS_ANGELES(13, 11:00)->TAMPA
T6:	TAMPA(13, 16:00)->LOS_ANGELES	LOS_ANGELES(13, 22:00)->TAMPA

where each line starts with the next move that the aircraft is scheduled to service (e.g., move T5). Following the move is the aircraft's current location (departure time from current location in the format of "date:time") and origin of the move, and then origin of the move (departure time from origin) and destination of the move.

The numerical example in the following section illustrates the use of this computer program for solving the operations planning problem with 10 aircraft and 6 moves.

8.4. A numerical example

We are given the following information for planning the aircraft operations involving 10 aircraft, 6 moves, and 7 different locations (including home bases, origins and destinations):

Aircraft	Home base
B747-P	TAMPA
C141	TAMPA
C130	TAMPA
C141	TAMPA
C5	BOSTON
LRWC	BOSTON
B747-P2	BOSTON
KC10-a	HOUSTON
KC10-b	HOUSTON
C5	HOUSTON

Move	Origin	Destination	Release-time	Due-time	Passenger	Bulk	Oversize	Outsize	Priority
T1	BOSTON	TAMPA	10	12	212	101	2002	0	1.2
T2	BOSTON	CHICAGO	1	3	77	166	722	0	0.5
T3	CHICAGO	LOS_ANGELES	1	2	474	0	0	0	2.0
T4	HONOLULU	ANCHORAGE	1	10	0	157	632	41	0.1
T5	CHICAGO	HOUSTON	7	9	41	107	0	0	1.0
T6	LOS_ANGELES	TAMPA	12	15	197	350	1978	138	1.1

Aircraft	Passenger	Bulk	Oversize	Outsize	Speed	Load-time	Unload-time	Max-flying-time
B747-P1	401	0	0	0	500	27 (min.)	27 (min.)	40 (hrs)
C5	73	83	72	78	450	160	150	40
C141	22	30	30	0	425	270	270	40
KC10	16	53	53	0	480	340	200	40
C130	8	14	13	0	267	270	270	40
LRWC	0	90	90	0	455	270	270	40

Home base/origin/destination location	Longitude	Latitude
BOSTON	-71090310	42319644
TAMPA	-82464687	27971159
CHICAGO	-87681192	41844044
LOS_ANGELES	-118387073	34099652
HONOLULU	-157830786	21314018
HOUSTON	-95387583	29761758
ANCHORAGE	-149761191	61194771

The computer print out, which specifies the routes constructed by our sequential assignment method, is as follows:

B747-P Base: TAMPA

```
T5: TAMPA(1, 0:00)->CHICAGO          CHICAGO(8, 0:00)->HOUSTON
T6: HOUSTON(8, 1:00)->LOS_ANGELES    LOS_ANGELES(13, 0:00)->TAMPA
T6: TAMPA(13, 5:00)->LOS_ANGELES     LOS_ANGELES(13, 11:00)->TAMPA
T6: TAMPA(13, 16:00)->LOS_ANGELES    LOS_ANGELES(13, 22:00)->TAMPA
```

C141 Base: TAMPA

```
T1: TAMPA(1, 0:00)->BOSTON           BOSTON(11, 4:00)->TAMPA
T1: TAMPA(11, 11:00)->BOSTON         BOSTON(11, 17:00)->TAMPA
T6: TAMPA(12, 0:00)->LOS_ANGELES     LOS_ANGELES(13, 4:00)->TAMPA
T6: TAMPA(13, 14:00)->LOS_ANGELES    LOS_ANGELES(13, 23:00)->TAMPA
T6: TAMPA(14, 9:00)->LOS_ANGELES     LOS_ANGELES(14, 18:00)->TAMPA
```

C130 Base: TAMPA

```
T3: TAMPA(1, 0:00)->CHICAGO          CHICAGO(2, 4:00)->LOS_ANGELES
T2: LOS_ANGELES(2, 17:00)->BOSTON    BOSTON(3, 9:00)->CHICAGO
T5: CHICAGO(3, 18:00)->CHICAGO       CHICAGO(8, 4:00)->HOUSTON
T5: HOUSTON(8, 12:00)->CHICAGO       CHICAGO(8, 19:00)->HOUSTON
```

C141 Base: TAMPA

```
T3: TAMPA(1, 0:00)->CHICAGO          CHICAGO(2, 4:00)->LOS_ANGELES
T4: LOS_ANGELES(2, 14:00)->HONOLULU  HONOLULU(3, 0:00)->ANCHORAGE
T4: ANCHORAGE(3, 11:00)->HONOLULU    HONOLULU(3, 21:00)->ANCHORAGE
```

C5 Base: BOSTON

```
T2: {waiting}                        BOSTON(2, 2:00)->CHICAGO
T3: CHICAGO(2, 7:00)->CHICAGO        CHICAGO(2, 9:00)->LOS_ANGELES
T2: LOS_ANGELES(2, 16:00)->BOSTON    BOSTON(3, 2:00)->CHICAGO
T2: CHICAGO(3, 6:00)->BOSTON         BOSTON(3, 11:00)->CHICAGO
T2: CHICAGO(3, 15:00)->BOSTON        BOSTON(3, 20:00)->CHICAGO
T6: CHICAGO(4, 0:00)->LOS_ANGELES    LOS_ANGELES(13, 2:00)->TAMPA
```

LRWC Base: BOSTON

```
T2: {waiting}                        BOSTON(2, 4:00)->CHICAGO
T3: CHICAGO(2, 11:00)->CHICAGO       CHICAGO(2, 15:00)->LOS_ANGELES
T4: LOS_ANGELES(3, 0:00)->HONOLULU   HONOLULU(3, 10:00)->ANCHORAGE
T4: ANCHORAGE(3, 21:00)->HONOLULU    HONOLULU(4, 7:00)->ANCHORAGE
```

(continued)

B747-P Base: BOSTON

T2: {waiting}	BOSTON(2, 0:00)->CHICAGO
T5: CHICAGO(2, 2:00)->CHICAGO	CHICAGO(8, 0:00)->HOUSTON
T5: HOUSTON(8, 1:00)->CHICAGO	CHICAGO(8, 3:00)->HOUSTON
T5: HOUSTON(8, 4:00)->CHICAGO	CHICAGO(8, 6:00)->HOUSTON
T5: HOUSTON(8, 7:00)->CHICAGO	CHICAGO(8, 9:00)->HOUSTON
T5: HOUSTON(8, 10:00)->CHICAGO	CHICAGO(8, 12:00)->HOUSTON
T1: HOUSTON(8, 13:00)->BOSTON	BOSTON(11, 0:00)->TAMPA
T1: TAMPA(11, 2:00)->BOSTON	BOSTON(11, 5:00)->TAMPA
T1: TAMPA(11, 7:00)->BOSTON	BOSTON(11, 10:00)->TAMPA
T1: TAMPA(11, 12:00)->BOSTON	BOSTON(11, 15:00)->TAMPA
T1: TAMPA(11, 17:00)->BOSTON	BOSTON(11, 20:00)->TAMPA
T1: TAMPA(11, 22:00)->BOSTON	BOSTON(12, 0:00)->TAMPA
T1: TAMPA(12, 3:00)->BOSTON	BOSTON(12, 5:00)->TAMPA

KC10 Base: HOUSTON

T3: HOUSTON(1, 0:00)->CHICAGO	CHICAGO(2, 5:00)->LOS_ANGELES
T1: LOS_ANGELES(2, 13:00)->BOSTON	BOSTON(11, 5:00)->TAMPA
T1: TAMPA(11, 11:00)->BOSTON	BOSTON(11, 18:00)->TAMPA
T1: TAMPA(12, 0:00)->BOSTON	BOSTON(12, 7:00)->TAMPA
T1: TAMPA(12, 13:00)->BOSTON	BOSTON(12, 20:00)->TAMPA
T6: TAMPA(13, 2:00)->LOS_ANGELES	LOS_ANGELES(13, 12:00)->TAMPA

KC10 Base: HOUSTON

T4: HOUSTON(1, 0:00)->HONOLULU	HONOLULU(2, 5:00)->ANCHORAGE
T4: ANCHORAGE(2, 14:00)->HONOLULU	HONOLULU(3, 0:00)->ANCHORAGE
T4: ANCHORAGE(3, 9:00)->HONOLULU	HONOLULU(3, 19:00)->ANCHORAGE

C5 Base: HOUSTON

T6: HOUSTON(1, 0:00)->LOS_ANGELES	LOS_ANGELES(13, 2:00)->TAMPA
T6: TAMPA(13, 10:00)->LOS_ANGELES	LOS_ANGELES(13, 17:00)->TAMPA
T6: TAMPA(14, 1:00)->LOS_ANGELES	LOS_ANGELES(14, 9:00)->TAMPA
T6: TAMPA(14, 16:00)->LOS_ANGELES	LOS_ANGELES(15, 0:00)->TAMPA

9. Remarks on implementing the results

For the purpose of implementing any of the results presented in this report, the following team members may be contacted for helping with the technical details:

Lei Lei — Office: (973)353-5185
Email: llei @andromeda.rutgers.edu

Zhiying Jin — (Sequential assignment method)
Office: (781)466-2567
Email: zjin@gte.com

Lijie Shi — (Column generation procedure)
Office: (201)788-6627
Email: SHIL@OMBLAN.CI.NYC.NY.US

REFERENCES

Ahuja, R. K., T. L. Magnanti and J. B. Orlin (1993), *Network Flows*, Prentice Hall.

Ball, M. O. et al (1995), *Handbooks in OR & MS*, Vol. 8, Elsevier Science.

Bodin, L., B. Golden, A. Assad, and M. Ball (1983), Routing and scheduling of vehicles and crews - the state of the art, *Computers & Operations Research*, 10, 63-211.

Bodin, L and T. Sexton (1986), The multi-vehicle subscriber dial-and-ride problem, *TIMS Studies Management Sciences*, 26, 73-86.

Dell'Amico, M., M. Fischetti and P. Toth (1993), Heuristic algorithms for the multiple deport vehicle scheduling problem, *Management Science*, 39, 1, 115-125.

Desaulniers, G., J. Lavigne, F. Soumis (1996), Multi-depot behicle scheduling problems with time windows and waiting cost, *Les Cahiers du, GERAD, issn: 0711-2440.*

Desrosiers, J., F. Soumis and M. Desrochers (1984), Routing with time windows by column generation, *Networks*, 14, 545-565.

Desrosiers, M. and F. Soumis (1988), A generalized permanent labeling algorithm for the shortest path problem with time windows, *INFOR*, 26(3), 191-212.

Dumas, Y., J. Desrosiers and F. Soumis (1991), The pick-up and delivery problems with time windows, *European Journal of Operations Research*, 54, 7-22.

Hooker, J. N. and N. R. Natraj (1995), Solving a general routing and scheduling problem by chain decomposition and tabu search, *Transportation Science*, 27, February, 30-44.

Gelinas, S., M. Desrochers, J. Desrosiers and M. Solomon (1995), A new branching strategy for time constrained routing problems with application to backhauling, *Annals of Operations Research*, 61, 91-109.

Golden, B. L., G. Laporte And E. D. Taillard (1997), An adaptive memory heuristic for a class of vehicle routing problems with minmax objective, Computers and Operations Research, 24, 5, 445-452.

Jaw, J., A. Odoni, H. Psaraftis and N. Wilson (1986), A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows, *Transportation Science*, 20B, 243-257.

Kohl, N. and O. B. G. Madsen (1995), An optimization algorithm for the vehicle routing problem with time windows based on Lagrangean relaxation, Working paper, Institute of

Mathematical Modeling, The Technical University of Denmark, DK-2800 Lyngby, Denmark.

Laporte, G. and Y. Norbert (1987), Exact algorithms for the vehicle routing problem, *Annals of Discrete Mathematics*, 31, 147-184.

Laporte, G., M. Desrochers and Y. Norbert (1984), Two exact algorithms for the distance-constrained vehicle routing problem, Networks, 14, 161-172.

Li, C. L., D. Simchi-Levi and M. Desrochers (1992), On the distance constrained vehicle routing problem, *Operations Research*, 40, 4, 1992, 790-799.

Potvin, J., T. Kervahut, B. L. Garcia and J. M. Rousseau (1996), The vehicle routing problem with time windows Part I: Tabu search, *INFORMS Journal on Computing*, 8, 2, 158-164.

Rappoport, H. K., L. S. Levy, B. L. Golden and K. Toussaint (1992), A planning heuristic for military airlift, *Interfaces*, 22, 73-87.

Roy, S., J. M. Rousseau, G&. Lapalme and J. A. Ferland (1984a), Routing and dispatching for the disabled persons - the algorithm, *Working paper TP5596E*, Transport Canada, Transport Development Center, Montreal.

Roy, S., J. M. Rousseau, G&. Lapalme and J. A. Ferland (1984b), Routing and dispatching for the disabled persons - the test, *Working paper TP5598E*, Transport Canada, Transport Development Center, Montreal.

Savelsbergh, M. W. P. (1984), Local search in routing problems with time windows, *Report OS-R8409*, Center for Mathematics and Computer Science, Amsterdam.

Solomon, M. M. (1986), On the worst case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints, *Networks*, 16, 161-174.

Solomon, M. M., E. K. Baker, J. Schaffer (1988), Vehicle routing and scheduling problems with time window constraints: efficient implementation of solution improvement procedures, *Vehicle Routing: Methods and Studies*, (edited by B. L. Golden and A. A. Assad), Elsevier Science Publishers, North-Holland.

APPENDIX Source code for Sequential assignment method

{The following program is coded in C++}

Input file: Aircraft.data

```
#Name Home
B747-P TAMPA
C141 TAMPA
C130 TAMPA
C141 TAMPA
C5 BOSTON
LRWC BOSTON
B747-P BOSTON
KC10 HOUSTON
KC10 HOUSTON
C5 HOUSTON
```

Input file: Aircraft.types

```
#Name Passenger Bulk Oversize Outsize Rate(speed) Load-time Unload-time
Max-flight-time
B747-P 401 0 0 0 500 27 27 40
C5 73 83 72 78 450 160 150 40
C141 22 30 30 0 425 270 270 40
KC10 16 53 53 0 480 340 200 40
C130 8 14 13 0 267 270 270 40
LRWC 0 90 90 0 455 270 270 40
```

Input file: Move.data

```
#Move Type Origin Destination Release-Time Due-Time Passenger Bulk
Oversize Outsize Priority(not zero)
T1 BOSTON TAMPA 10 12 212 101 2002 0 1.2
T2 BOSTON CHICAGO 1 3 77 166 722 0 0.5
T3 CHICAGO LOS_ANGELES 1 2 474 0 0 0 2.0
T4 HONOLULU ANCHORAGE 1 10 0 157 632 41 0.1
T5 CHICAGO HOUSTON 7 9 41 107 0 0 1.0
T6 LOS_ANGELES TAMPA 12 15 197 350 1978 138 1.1
```

Input file: Location.xy

```
#Name Longitude Latitude
BOSTON -71090310 42319644
TAMPA -82464687 27971159
CHICAGO -87681192 41844044
LOS_ANGELES -118387073 34099652
HONOLULU -157830786 21314018
HOUSTON -95387583 29761758
ANCHORAGE -149761191 61194771
```

Header file

```
/*****************************************************************/
/*                          air_schedule.h                       */
/*****************************************************************/
#ifndef AIR_SCHEDULE_H
#define AIR_SCHEDULE_H

#define INFINITY        1073741823  /* (2 to the power 30) -1 */
#define BIG_M           INFINITY
```

```
#define INT_malloc(N)     ((int *)(malloc((N)*sizeof(int))))
#define NET_malloc(N)    ((Farc **)(malloc((N + 1) * sizeof(Farc *))))
#define B_malloc(N)      ((Barc **)(malloc((N + 1) * sizeof(Barc *))))

#define AIRCRAFT_malloc(N)    ((Aircraft *)(malloc((N)*sizeof(Aircraft
))))
#define MOVE_malloc(N)    ((Move *)(malloc((N)*sizeof(Move ))))

#define MILES_PER_UNIT 0.000069

typedef struct arc_entry{
  int cost,                      /* cost of arc */
      tail,                      /* tail of barc */
      head;                      /* head of arc */
  int arrive_time, travel_time, leave_time;
  struct arc_entry *next;        /* next arc on the arc-list */
}Farc, Arc;

typedef struct barc_entry{
  struct arc_entry *arc;         /* arc of the barc */
  struct barc_entry *next;       /* next barc on the barc-list */
}Barc;

typedef struct solution_struct {
  int start_day, start_hour;
  int leave_day, leave_hour;
  int from_index;
  int move_index;
  struct solution_struct *next;
}Solution_Struct;

typedef struct aircraft_entry{
  int type_index, location_index;
  int home_index;
  int start_time, travel_time;
  struct Solution_Struct *first_solution, *cur_solution;
}Aircraft;

typedef struct move_entry{
  char type[32];
  int passenger, bulk, oversize, outsize;
  int origin_index, destination_index;
  int release_time, due_time;
  double priority;
  int stayed;
}Move;

typedef struct aircraft_type {
  char type[16];
  int passenger, bulk, oversize, outsize, speed, load_time, unload_time,
flight_time;
}Aircraft_type;

typedef struct loaction {
  char name[32];
  int x, y;
}Location;


/*****************************************************************/
```

```
class Assignment {
 private:
     Farc **first_arc, **pre_x, **pre_y, **y;
     Barc **first_barc;
     int **cost, *u, *v, *d, *free_nodes, *cols, *queue;
     int check_early_time, round;

     int CheckConstraints(int i, int j, int &cost, int &arrive_time, int
&travel_time, int &leave_time);
     int Solution();
     int ConstructNetwork();
     int Update();
     int FreeNetwork();
     void OutputSolution();

 public:
     double obj;
     int *x;
     Aircraft *aircraft;
     Move *move;
     Aircraft_type *aircraft_type;
     Location *location;
     int num_rows, num_cols, num_types, num_locations, max_num_rows,
max_num_cols, num_arcs;
     Assignment(int max_num_aircrafts, int max_num_moves, int
max_num_types, int max_num_locations);
     int FindSolution();
     ~Assignment();
};

#endif
```

Source code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "air_schedule.h"

#define TRUE            1
#define FALSE           0

/*************************************************************************/
Assignment::Assignment(int max_num_aircraft, int max_num_move, int
max_num_types, int max_num_locations)
{
   max_num_rows = max_num_aircraft;
   max_num_cols = max_num_move;

     /* allocating memory */

  int status = 0;
  if(!(x                = INT_malloc(max_num_rows + max_num_cols + 1)))
status = 1;
  else if(!(y           = NET_malloc(max_num_cols + 1))) status = 1;
  else if(!(u           = INT_malloc(max_num_rows + 1))) status = 1;
  else if(!(v           = INT_malloc(max_num_cols + 1))) status = 1;
  else if(!(free_nodes = INT_malloc(max_num_cols + 1))) status = 1;
```

```
  else if(!(cols        = INT_malloc(max_num_cols + 1))) status = 1;
  else if(!(pre_y        = NET_malloc(max_num_cols + 1))) status = 1;
  else if(!(pre_x        = NET_malloc(max_num_rows + 1))) status = 1;
  else if(!(d          = INT_malloc(max_num_cols + 1))) status = 1;
  else if(!(queue          = INT_malloc(max_num_cols + max_num_rows +
1))) status = 1;
  else if (!(first_arc = NET_malloc(max_num_rows + 1))) status = 1;
  else if (!(first_barc = B_malloc(max_num_cols + 1))) status = 1;


  /* see if there is any error in allocating memory */
  if(status != 0) {
    free(x); free(y); free(u); free(v); free(cols); free(pre_x);
free(pre_y);
    free(d); free(free_nodes); free(queue); free(first_arc);
free(first_barc);
    printf("Error in allocating memories\n");
    exit(1);
  }

  if (!(aircraft = AIRCRAFT_malloc(max_num_aircraft + 1))) {
     printf("Cann't allocate memories for aircraft %d\n",
max_num_aircraft);
     exit(1);
  }

  if (!(move = MOVE_malloc(max_num_move + 1))) {
     printf("Cann't allocate memories for moves %d\n", max_num_move);
     exit(1);
  }

  char *filename;
  char buf[2560];
  FILE *fp;

  /* load aircraft types file */
  filename = "aircraft.types";
  if (!(fp = fopen(filename, "r"))) {
     printf("Cann't open file %s\n", filename);
     exit (1);
   }
  aircraft_type = (Aircraft_type *) malloc(max_num_types *
sizeof(Aircraft_type));
  num_types = 0;
  while (fgets(buf, sizeof(buf), fp)) {
     if (buf[0] == '#') continue;
     if (num_types >= max_num_types) continue;
     sscanf(buf, "%s %d %d %d %d %d %d %d %d",
aircraft_type[num_types].type,

&(aircraft_type[num_types].passenger),
                                  &(aircraft_type[num_types].bulk),
                                  &(aircraft_type[num_types].oversize),
                                  &(aircraft_type[num_types].outsize),
                                  &(aircraft_type[num_types].speed),

&(aircraft_type[num_types].load_time),

&(aircraft_type[num_types].unload_time),

&(aircraft_type[num_types].flight_time));
```

```
    aircraft_type[num_types].flight_time = 60 *
aircraft_type[num_types].flight_time;
    num_types ++;
   }
  fclose(fp);

  printf("Done with file %s\n", filename);

  /* load location file */
  filename = "location.xy";
  if (!(fp = fopen(filename, "r"))) {
     printf("Cann't open file %s\n", filename);
     exit (1);
   }
  location = (Location *) malloc(max_num_locations * sizeof(Location));
  num_locations = 0;
  while (fgets(buf, sizeof(buf), fp)) {
     if (buf[0] == '#') continue;
     if (num_locations >= max_num_locations) continue;
     sscanf(buf, "%s %d %d", location[num_locations].name,
                             &(location[num_locations].x),
                             &(location[num_locations].y));
     num_locations ++;
   }
  fclose(fp);

  printf("Done with file %s\n", filename);

  /* load aircraft data */
  filename = "aircraft.data";
  if (!(fp = fopen(filename, "r"))) {
     printf("Cann't open file %s\n", filename);
     exit (1);
   }
  num_rows = 0;
  char type[32], location_name[32];
  int i, type_index, location_index;
  while (fgets(buf, sizeof(buf), fp)) {
     if (buf[0] == '#') continue;
     if (num_rows >= max_num_rows) continue;
     sscanf(buf, "%s %s", type, location_name);
     num_rows ++;
     for (i = 0; i < num_types; i ++) {
         if (strcmp(type, aircraft_type[i].type) == 0) break;
       }
     if (i >= num_types) {
        printf("Cann't find type %s\n", type);
        continue;
      }
     type_index = i;
     for (i = 0; i < num_locations; i ++) {
         if (strcmp(location_name, location[i].name) == 0) break;
       }
     if (i >= num_locations) {
        printf("Cann't find location %s\n", location_name);
        continue;
      }
     location_index = i;
     aircraft[num_rows].type_index = type_index;
     aircraft[num_rows].location_index = aircraft[num_rows].home_index =
location_index;
```

```
      aircraft[num_rows].start_time = aircraft[num_rows].travel_time = 0;
      first_arc[num_rows] = NULL;
      aircraft[num_rows].first_solution = NULL;
    }
  fclose(fp);

  printf("Done with file %s\n", filename);

  /* load move data */
  filename = "move.data";
  if (!(fp = fopen(filename, "r"))) {
     printf("Cann't open file %s\n", filename);
     exit (1);
    }
  char from_loc[32], to_loc[32], move_type[16];
  int passenger, bulk, oversize, outsize, release_time, due_time;
  double priority;
  int from_loc_index, to_loc_index;
  num_cols = 0;
  while (fgets(buf, sizeof(buf), fp)) {
     if (buf[0] == '#') continue;
     if (num_cols >= max_num_cols) continue;
     sscanf(buf, "%s %s %s %d %d %d %d %d %d %lf", move_type, from_loc,
to_loc,
                                          &release_time, &due_time,
                                          &passenger, &bulk,
                                          &oversize, &outsize,
&priority);
     for (i = 0; i < num_locations; i ++) {
         if (strcmp(from_loc, location[i].name) == 0) break;
       }
     if (i >= num_locations) {
        printf("Cann't find location %s\n", from_loc);
        continue;
      }
     from_loc_index = i;
     for (i = 0; i < num_locations; i ++) {
         if (strcmp(to_loc, location[i].name) == 0) break;
       }
     if (i >= num_locations) {
        printf("Cann't find location %s\n", to_loc);
        continue;
      }
     num_cols ++;
     to_loc_index = i;
     move[num_cols].origin_index = from_loc_index;
     move[num_cols].destination_index = to_loc_index;
     move[num_cols].passenger = passenger;
     move[num_cols].bulk = bulk;
     move[num_cols].oversize = oversize;
     move[num_cols].outsize = outsize;
     move[num_cols].release_time = release_time * 24 * 60;
     move[num_cols].due_time = due_time * 24 * 60;
     move[num_cols].priority = priority;
     sprintf(move[num_cols].type, move_type);
     first_barc[num_cols] = NULL;
   }
  fclose(fp);

  printf("Done with file %s\n", filename);
```

```
}

/*************************************************************************/
Assignment::~Assignment()
{
  FreeNetwork();
}

/*-----------------------------------------------------------------------*/
int Assignment::FindSolution()
{
  obj = 0;
  round = 1;
  int find = ConstructNetwork();
  while (find) {
          Solution();
        Update();
          find = ConstructNetwork();
          round ++;
  }

  OutputSolution();

  return TRUE;
}

/*************************************************************************/
void Assignment::OutputSolution()
{

  char *filename = "output.txt";
  FILE *fp_out;

  if (!(fp_out = fopen(filename, "w"))) {
     printf("Cann't open file %s\n", filename);
     fp_out = NULL;
   }

  int type_index, j, start_day, start_hour, leave_day, leave_hour;
  int location_index;

  Solution_Struct *solution, *solution1;
  for (int i = 1; i <= num_rows; i ++) {
      if (aircraft[i].first_solution == NULL) continue;
      solution = aircraft[i].first_solution;
      type_index = aircraft[i].type_index;
      if (fp_out) {
         fprintf(fp_out, "    %s Base: %s\n",
aircraft_type[type_index].type,

location[aircraft[i].home_index].name);
         fprintf(fp_out, "-----------------------------------------------
------------\n");
       }
      while (solution) {
            start_day = solution->start_day;
            start_hour = solution->start_hour;
            leave_day = solution->leave_day;
            leave_hour = solution->leave_hour;
            location_index = solution->from_index;
            j = solution->move_index;
```

```
            if (fp_out) {
               fprintf(fp_out, "%s: %s(%d, %d:00)->%s    %s(%d, %d:00)-
>%s\n",
                     move[j].type,
                     location[location_index].name,
                     start_day+1, start_hour,
                     location[move[j].origin_index].name,
                     location[move[j].origin_index].name,
                     leave_day+1, leave_hour,
                     location[move[j].destination_index].name);
            }
             solution1 = solution;
             solution = solution->next;
             free (solution1);
         }
        if (fp_out) fprintf(fp_out, "\n");
    }

    if (fp_out) fclose(fp_out);
}

/*************************************************************************/
int Assignment::Update()
{
      Farc *arc;
        int i, j, type_index, location_index, start_day, start_hour,
leave_day, leave_hour;
        Solution_Struct *solution;

      for (i = 1; i < num_rows + 1; i++) {
          if (x[i] == 0) continue;
            j = x[i];
          arc = y[j];
            type_index = aircraft[i].type_index;
            location_index = aircraft[i].location_index;
            start_day = aircraft[i].start_time/(24*60);
            start_hour = (aircraft[i].start_time)/60 - 24 * start_day;
            leave_day = arc->leave_time/(24*60);
            leave_hour = arc->leave_time/60 - 24 * leave_day;
            solution = (Solution_Struct *)
malloc(sizeof(Solution_Struct));
            solution->start_day = start_day;
            solution->start_hour = start_hour;
            solution->from_index = location_index;
            solution->leave_day = leave_day;
            solution->leave_hour = leave_hour;
            solution->move_index = j;
            solution->next  = NULL;
            if (aircraft[i].first_solution == NULL)
               aircraft[i].first_solution = solution;
            else
               aircraft[i].cur_solution->next = solution;
            aircraft[i].cur_solution = solution;
            /* location */
            aircraft[i].location_index = move[j].destination_index;
            aircraft[i].start_time = arc->arrive_time +
aircraft_type[type_index].unload_time;
            aircraft[i].travel_time = aircraft[i].travel_time + arc-
>travel_time;
            move[j].passenger = move[j].passenger -
aircraft_type[type_index].passenger;
```

```
            move[j].bulk = move[j].bulk -
aircraft_type[type_index].bulk;
            move[j].oversize = move[j].oversize -
aircraft_type[type_index].oversize;
            move[j].outsize = move[j].outsize -
aircraft_type[type_index].outsize;
        }

        return (TRUE);
}

/*-------------------------------------------------------------------------*/
int Assignment::Solution()
{
  int f, f0, up, low, aug,
      c, min_cost, u1, u2, z, uu,
      i, j, k, t, i1, j1, count;
  Farc  *arc, *arc1, *arc2, *marc;
  Barc *barc, *barc1;

  /* empty row matches */
  for(i=1;i<=num_rows;i++) x[i] = 0;

  for(j=1;j<=num_cols;j++) y[j]    = NULL;

  /* column reduction */
  for(j=num_cols; j>0; j--){
    barc1 = first_barc[j];
    if (!barc1) continue;
    z     = barc1->arc->cost;
    barc  = barc1->next;
    while(barc){
      if((c = barc->arc->cost) < z){
          z     = c;
          barc1 = barc;
      }
      barc = barc->next;
    }
    v[j] = z;
    i1 = barc1->arc->tail;
    if(x[i1] == 0){
      y[j] = barc1->arc;
      x[i1] = j;
    }
  }

  /* reduction transfer */
  f = 0;
  for(i=1; i<=num_rows; i++)
    if(x[i] == 0) free_nodes[++f] = i;

  /* shortest-path augmentation */
  f0 = f; f=0;
  for(t=1; t<=f0; t++){
      for (j = 1; j <= num_cols; j ++) {
        d[j] = INFINITY;
        queue[j] = FALSE;
      }
    i1  = free_nodes[t];
      aug = 0;
      min_cost = INFINITY - 1;
```

```
  low = 0;
  up = num_cols;
arc = first_arc[i1];
while(arc){
       j      = arc->head;
       d[j]   = arc->cost - v[j];
       pre_y[j] = arc;
     if (d[j] <= min_cost) {
        if (d[j] < min_cost) {
         low = 0;
             min_cost = d[j];
         }
        low ++;
        cols[low] = j;
      }
       arc    = arc->next;
  }

 if (low == 0) continue;

 for (k = 1; k <= low; k ++) {
   j = cols[k];
   if (y[j] == NULL) {
      aug = 2;
      break;
    }
   queue[j] = TRUE;
   }

if (aug == 0) {
do{
        j = cols[low];
        low --;
        arc = y[j];
        cols[up] = j;
        up --;
        uu = arc->cost - v[j] - min_cost;
        arc = first_arc[arc->tail];
        while (arc) {
                  j = arc->head;
              if (queue[j] == FALSE) {
                    z = arc->cost - v[j] - uu;
                    if (z < d[j]) {
                          d[j] = z;
                          pre_y[j] = arc;
                          if (z == min_cost) {
                                if (y[j] == NULL) {
                                      aug = 1;
                                      break;
                                }
                                low ++;
                                cols[low] = j;
                                queue[j] = TRUE;
                          }
                    }
              }
              arc = arc->next;
        }

        if (aug) break;
        if (low == 0) {
```

```
                min_cost = INFINITY - 1;
                for (j = 1; j <= num_cols; j++) {
                      if (queue[j] == FALSE && d[j] <= min_cost) {
                            if (d[j] < min_cost) {
                                  low = 0;
                                  min_cost = d[j];
                            }
                            low ++;
                            cols[low] = j;
                      }
                }
                for (k = 1; k <= low; k++) {
                      j = cols[k];
                      if (y[j] == NULL) {
                            aug = 1;
                            break;
                      }
                      queue[j] = TRUE;
                }
          }
          if (aug) break;
          if (low == 0) break;
     } while (1 == 1);
  }

    /* augmentation */
    if(aug == 1){
      for(k=up + 1; k<=num_cols; k++){
           j1    = cols[k];
           v[j1] = v[j1] + d[j1] - min_cost;
      }
    }
    if (aug) {
      do {
         arc = pre_y[j];
         y[j] = arc;
         i = arc->tail;
         k = j;
         j = x[i];
         x[i] = k;
       } while (i != i1);
    }
    else f++;
  }

  for (i = 1; i <= num_rows; i ++) {
      j = x[i];
      if (j != 0) {
         u[i] = y[j]->cost - v[j];
         obj = obj + u[i] + v[j];
       }
      else u[i] = 0;
    }

  if(f) return(-1);
  else  return(0);
}

/*****************************************************************/
int Assignment::ConstructNetwork()
{
```

```
  int i, j, cost, arrive_time, travel_time, leave_time;
  int find;
  Farc *arc, *next_arc;
  Barc *barc, *next_barc;

  for(i=1; i<num_rows+1; i++) {
     arc = first_arc[i];
     while (arc) {
         next_arc = arc->next;
         free (arc);
         arc = next_arc;
       }
     first_arc[i]  = NULL;
   }

  for(j=1; j<num_cols+1; j++) {
     barc = first_barc[j];
     while (barc) {
         next_barc = barc->next;
         free (barc);
         barc = next_barc;
       }
     first_barc[j] = NULL;
   }

  find = FALSE;

  /* find max start time */
  int max_start_time = 0;
  for (i = 1; i < num_rows+1; i ++)
      if (aircraft[i].start_time > max_start_time) max_start_time =
aircraft[i].start_time;

  /* find min release time */
  int min_release_time = move[1].release_time;
  for (j = 1; j < num_cols + 1; j ++)
      if (move[j].release_time < min_release_time) min_release_time =
move[j].release_time;

  if (min_release_time > (max_start_time - 24 * 60)) check_early_time =
FALSE;
  else check_early_time = TRUE;

  for (i = 1; i < num_rows+1; i++) {
    for (j = 1; j < num_cols+1; j++) {
      if (CheckConstraints(i, j, cost, arrive_time, travel_time,
leave_time)) {
           if (!(arc = (Farc *)(malloc(sizeof(Farc))))) return FALSE;
           find = TRUE;
           arc->tail = i;
           arc->head = j;
           arc->cost = cost;
           arc->arrive_time = arrive_time;
           arc->travel_time = travel_time;
           arc->leave_time = leave_time;
           arc->next = first_arc[i];
           first_arc[i] = arc;
           num_arcs++;
           if (!(barc        = (Barc *)(malloc(sizeof(Barc))))) return
FALSE;
           barc->arc   = arc;
```

```
            barc->next  = first_barc[j];
            first_barc[j] = barc;
          }
        }
    }

    return (find);
}

/*************************************************************************/
int Assignment::CheckConstraints(int row, int column, int &cost, int
&arrive_time_, int &travel_time_, int &leave_time)
{
    if (move[column].passenger <= 0 && move[column].bulk <= 0 &&
        move[column].oversize <= 0 && move[column].outsize <= 0) return
FALSE;

    int travel_time1, travel_time2, travel_time3;

    int x1 = location[aircraft[row].location_index].x;
    int y1 = location[aircraft[row].location_index].y;
    int x2 = location[move[column].origin_index].x;
    int y2 = location[move[column].origin_index].y;
    int x3 = location[move[column].destination_index].x;
    int y3 = location[move[column].destination_index].y;

    int speed = aircraft_type[aircraft[row].type_index].speed;
    int travel_time = aircraft[row].travel_time;

    /* from start location to origin */
    double xx = x2 - x1;
    double yy = y2 - y1;
    xx = xx * xx;
    yy = yy * yy;
    double dist = sqrt(xx + yy) * MILES_PER_UNIT;
    travel_time1 = 60 * (int) (dist/(double) (speed));
    int arrive_time = aircraft[row].start_time + travel_time1;
    if (arrive_time < move[column].release_time) arrive_time =
move[column].release_time;
    if (check_early_time &&
        arrive_time < (move[column].release_time - 24 * 60))
       return FALSE;

    /* from origin to destination */
    xx = x2 - x3;
    yy = y2 - y3;
    xx = xx * xx;
    yy = yy * yy;
    dist = sqrt(xx + yy) * MILES_PER_UNIT;
    travel_time2 = 60 * (int) (dist/(double) (speed));
    leave_time = arrive_time +
aircraft_type[aircraft[row].type_index].load_time;
    arrive_time = arrive_time +
aircraft_type[aircraft[row].type_index].load_time + travel_time2;
    arrive_time_ = arrive_time;
    if (arrive_time > move[column].due_time) return FALSE;

    /* from destination to start location */
    xx = x3 - x1;
    yy = y3 - y1;
    xx = xx * xx;
```

```
  yy = yy * yy;
  dist = sqrt(xx + yy) * MILES_PER_UNIT;
  travel_time3 = 60 * (int) (dist/(double) (speed));
  travel_time = travel_time + travel_time1 + travel_time2 +
travel_time3;
  if (travel_time > aircraft_type[aircraft[row].type_index].flight_time)
     return FALSE;

  travel_time_ = travel_time1 + travel_time2;

  if (move[column].priority > 0.0)
     cost = (int) ((double) (travel_time1 +
travel_time2)/move[column].priority);
  else cost = travel_time1 + travel_time2;

  return TRUE;
}

/*************************************************************************/
int Assignment::FreeNetwork()
{
  int   i;
  Farc   *arc, *next_arc;
  Barc  *barc, *next_barc;

  free(y); free(u); free(v); free(cols); free(pre_x); free(pre_y);
  free(d); free(free_nodes); free(queue);

  if (x) free(x);
  if(first_barc){
    for(i=1; i<num_cols+1; i++){
      barc = first_barc[i];
      while(barc){
      next_barc = barc->next;
      free(barc);
      barc = next_barc;
      }
    }
    free(first_barc);
  }

  if(first_arc){
    for(i=1; i<num_rows+1; i++){
      arc = first_arc[i];
      while(arc){
              next_arc = arc->next;
              free(arc);
              arc = next_arc;
      }
    }
    free(first_arc);
  }

  if (aircraft) free(aircraft);
  if (move) free(move);
  if (aircraft_type) free(aircraft_type);
  if (location) free(location);

  return TRUE;

}
```

*U.S. GOVERNMENT PRINTING OFFICE: 2000-510-079-81262

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.